UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jaka Cikač

Primerjava algoritmov za porazdeljeno preiskovanje prostora v simulacijskem okolju

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Danijel Skočaj

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Porazdeljeno preiskovanje prostora je ena izmed pogostih nalog umeščenih ekip inteligentnih agentov. Preučite različne metode za porazdeljeno preiskovanje prostora, ki se pojavljajo na področju večagentnih sistemov. Implementirajte tri tipične predstavnike različnih razredov algoritmov in primerjajte njihovo delovanje. V ta namen razvijte ustrezno simulacijsko okolje, ki bo omogočalo vzporedno delovanje poljubnega števila agentov. V simulacijskem okolju na različnih zemljevidih eksperimentalno ovrednotite implementirane algoritme in ustrezno interpretirajte rezultate.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Jaka Cikač, z vpisno številko **63100181**, sem avtor diplomskega dela z naslovom:

Primerjava algoritmov za porazdeljeno preiskovanje prostora v simulacijskem okolju

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Danijela Skočaja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. septembra 2014 Podpis avtorja:

"Where other men blindly follow the truth, remember..."

"*Nothing is true.*"

"Where other men are limited by morality or law, remember..."

"*Everything is permitted.*"

"We work in the dark to serve the light. We are *Engineers.*"

Kazalo

Slike

Tabele

Algoritmi

Seznam uporabljenih kratic

kratica	angleško	slovensko
DAI	Distributed Artificial Intelligence	porazdeljena umetna inteligenca
AI	Artificial Intelligence	umetna inteligenca
MAS	Multi-Agent System	večagentni sistem
DPS	Distributed Problem Solving	porazdeljeno reševanje problemov
PAI	Parallel Artificial Intelligence	paralelna umetna inteligenca
SLAM	Simultaneous Localization And Mapping	sočasna lokalizacija in mapiranje prostora
PSO	Particle Swarm Optimisation	optimizacija roja delcev
dPSO	distributed Particle Swarm Optimisation	porazdeljena optimizacija roja delcev
RPSO	Robotic Particle Swarm Optimisation	robotska optimizacija roja delcev
DPSO	Darwinian Particle Swarm Optimisation	Darwinova optimizacija roja delcev
RDPSO	Robotic Darwinian Particle Swarm Optimisation	robotska Darwinova optimizacija roja delcev

Povzetek

Cilj algoritmov za preiskovanje prostora je odkriti čim več neodkritega prostora v čim krajšem času in čim bolj učinkovito. Da bi to dosegli, se poslužimo porazdeljenih algoritmov, ki jih uprabimo na večagentnih sistemih. V delu želimo odkriti, kateri izmed algoritmov lahko učinkovito preiščejo prostor v simulacijskem okolju Gridland. Ker okolje v originalni različici ni namenjeno preiskovanju prostora, je bilo okolje potrebno prilagoditi in omogočiti spremljanje zgodovine premikov ter akcij večagentnega sistema za kasnejšo analizo učinkovitosti algoritmov. Za referenčno oceno smo implementirali naključnega agenta, tega pa primerjali z algoritmom, ki zastopa skupino tako imenovanih "pseudo-naključnih" algoritmov in z algoritmom, ki temelji na optimizaciji roja delcev. Pokazali smo, da so pseudo-naključni algoritmi veliko boljši od naključnih, kljub njihovi enostavnosti. Algoritem RDPSO, ki temelji na optimizaciji roja delcev, pa se je izkazal za učinkovitega, čeprav ni najhitrejši.

Ključne besede: večagentni sistemi, optimizacija roja delcev, preiskovanje prostora, gridland.

Abstract

Space exploration algorithms aim to discover as much unknown space as possible as efficiently as possible in the shortest possible time. To achieve this goal, we use distributed algorithms, implemented on multi-agent systems. In this work, we explore, which of the algorithms can efficiently explore space in a simulated environment Gridland. Since Gridland, in it's original release, was not meant for simulating space exploration, we had to make some modifications and enable movement history and action tracking for a multi-agent system with the purpose of algorithm efficiency analysis. A random agent was implemented for reference and compared with an algorithm, that represents a group of so called "pseudo-random" algorithms, and a particle swarm based algorithm. We show that pseudo-random algorithms are much better than random algorithms, despite their simplicity. Algorithm RDPSO, based on particle swarm optimisation, proved to be efficient, despite not being the fastest.

Keywords: multi-agent systems, particle swarm optimisation, space exploration, gridland.

Poglavje 1

Uvod

1.1 Motivacija

Večina od nas ima potrebo po odkrivanju neznanega, iskanju novega znanja in željo po pustolovščini, ki jo raziskovanje prinese s seboj. Zdi se, da več kot odkrijemo, več ostaja neodkritega. Že dolgo časa raziskujemo naš planet, pa vendar še vedno ne vemo kaj vse skrivajo naši oceani [16]. Prav tako ostaja neraziskana velika večina podzemnih jam. Raziskovanje smo pomaknili tudi v vesolje in morda bomo kmalu podrobneje spoznali tudi druge planete našega Osončja. Odkrivanje novih delov oceana, jam in planetov zahteva veliko časa in je lahko za ljudi precej nevarno. Algoritmi za preiskovanje prostora lahko pripomorejo k odkrivanju neznanega in prihranijo ogromno časa in tveganja, če jih uporabimo na robotih, ki so sposobni raziskovanja neznanega terena.

S pomočjo inteligentnih sistemov si lahko prihranimo veliko časa in sredstev pri raziskovanju. Za še večjo učinkovitost se lahko poslužimo večagentnih sistemov, kjer medseboj povežemo več inteligentnih agentov oziroma robotov. Razlogi za medsebojno povezavo večih agentov so, da jim omogočimo

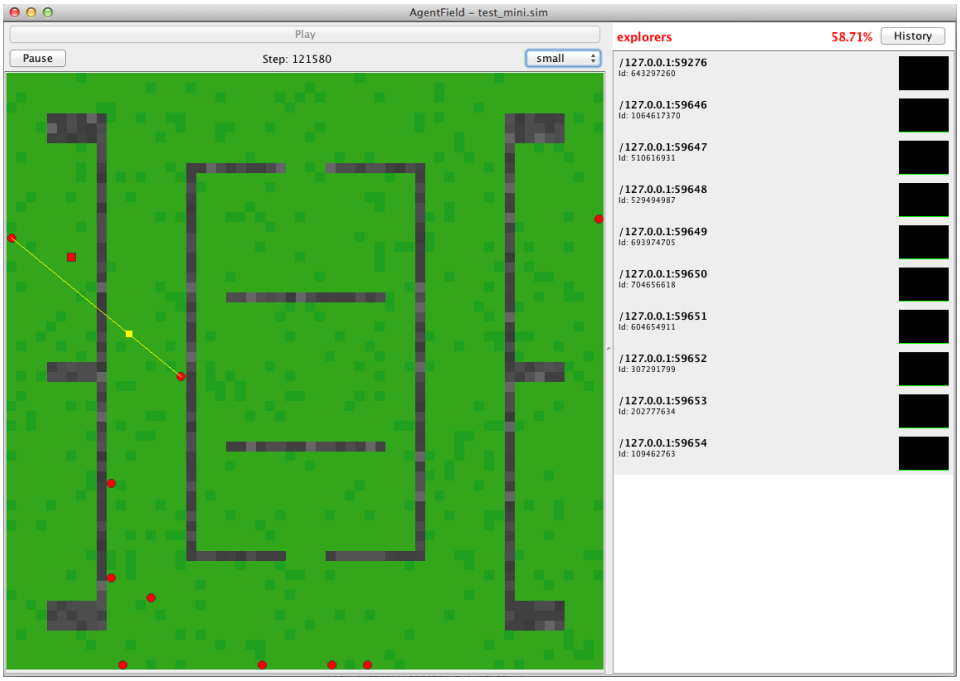
sodelovanje pri reševanju problemov z uporabo izmenjave znanja in vzporednega sodelovanja ter odpornost na napake zaradi redundance. Ker želimo, da so med seboj agenti avtonomni (tako preprečimo odpoved sistema zaradi napake na centralnem sistemu), agente razvijemo modularno in porazdeljeno.

Porazdeljene rešitve so ponavadi lažje za razvoj in razumevanje, če narava problema, ki ga rešujemo, omogoča porazdelitev. To lahko vodi do algoritmov in strategij, ki še niso bile odkrite s centraliziranim prijemom in so lahko veliko bolj učinkovite.

1.2 Cilj diplomske naloge

V zadnjih letih je bilo raziskovanje področja preiskovanja prostora s pomočjo večagentnih sistemih zelo aktivno in raziskovalci so razvili veliko algoritmov, ki učinkovito preiskujejo prostor ali iščejo cilje v prostoru. Razvoj se je razdelil v tri večje skupine, ki jih bomo v diplomskem delu podrobneje opisali. Nekatere algoritme bomo med seboj tudi primerjali.

Cilj algoritmov je preiskati čim več prostora, agentom neznanega, zemljevida v čim krajšem času in čim bolj učinkovito. S pomočjo simulacijskega okolja Gridland je agentom omogočena komunikacija in koordinacija njihovih akcij, da lahko cilj dosežejo hitreje in bolj učinkovito. Prav tako lahko agenti med seboj izmenjujejo zemljevid že obiskanega prostora. Primer simulacije večagentnega sistema, ki preiskuje prostor v simulacijskem okolju Gridland je prikazan na Sliki 1.1.



Slika 1.1: Prikaz simulacije v okolju Gridland.

1.3 Zgradba diplomske naloge

V diplomskem delu najprej predstavimo porazdeljeno umetno inteligenco in večagentne sisteme ter okolje, v katerem lahko sistem deluje. Nato naredimo pregled algoritmov, ki se uporabljajo v namene preiskovanja prostora z večagentnim sistemom in podrobneje predstavimo optimizacijo roja delcev. Sledi opis prilagoditve simulacijskega okolja Gridland, za katerega implementiramo naključnega agenta, "pseudo-agenta" in agenta RDPSO ter jih med seboj primerjamo. V zadnjem delu predstavimo rezultate in analizo učinkovitosti implementiranih algoritmov.

Poglavje 2

Porazdeljena umetna inteligenca in večagentni sistemi

2.1 Porazdeljena umetna inteligenca

Porazdeljena umetna inteligenca je področje umetne inteligence (UI, ang. "artificial intelligence" - AI), ki se ukvarja s koordinacijo, reševanjem problemov in sočasnimi akcijami. Deli se na tri večja področja - porazdeljeno reševanje problemov (ang. "distributed problem solving" - DPS), večagentni sistemi (ang. "multi-agent systems" - MAS) in paralelna umetna inteligenca (ang. "parallel artificial intelligence" - PAI)[11].

Pred centraliziranimi rešitvami imajo porazdeljene veliko prednosti, zahtevajo pa drugačen pristop k problemu, ki ga rešujemo.

Topologija porazdeljenih sistemov je dinamična in njihova vsebina se spreminja zelo hitro, zato uporabnik ali aplikacija ne more tako hitro pridobiti informacij in skrbeti za konsistentnost. Za obravnavo takih informacij, sistemov in problemov so na voljo štiri tehnike, to so modularnost, porazdeljenost,

abstrakcija in umetna inteligenca [19].

To so osnove za porazdeljeno umetno inteligenco (PUI, ang. "distributed artificial intelligence" - DAI).

Osnovni problemi porazdeljene umetne inteligence so [11]

- kako formulirati, opisati in razdeliti probleme ter kako združiti rezultate posameznih agentov,
- kako zagotoviti skladnost za odločanje in reagiranje z upoštevanjem posledic v globalnem pogledu,
- kako omogočiti komunikacijo in interakcijo med agenti,
- kako omogočiti posameznim agentom, da si predstavijo razum, akcije, načrte in znanje ostalih agentov za možnost koordinacije z njimi, ter kako razumeti stanje koordinacije.

2.1.1 Razvoj porazdeljenih sistemov

Pri razvoju sistemov PUI moramo upoštevati določene karakteristike, ki so potrebne za uspešno delovanje sistema, ki sledi principom porazdeljene umetne inteligence. Nekaj takih karakteristih je navedenih v Tabeli 2.1.

Pri razvoju sistema in okolja, moramo upoštevati, da so možne istočasne akcije (več agentov izvaja različne ali enake akcije v istem času) in se zavedati, da različni agenti morda stremijo k različnim ciljem (usmerjena pozornost). V nekaterih sistemih je možno, da so agenti med seboj različni, tako imajo lahko večjo avtoriteto in različno odgovornost. Agente se lahko tudi kaznuje ali nagrajuje z zaupanjem, glede na delo, ki ga opravijo v sistemu.

Tabela 2.1: Karakteristike sistema, povzeto po [19].

Prilagodljivost	sistem mora biti prilagodljiv na različne situacije
Cena enot	praviloma so posamezne enote enostavne, nizko cenovne in učinkovite pri komunikaciji
Pristop k razvoju	možnost razširitve sistema in nadaljnega razvoja
Učinkovitost	zaradi sočasnih akcij se hitrost sistema poveča, imamo pa obremenitev zaradi vzdrževanja koordinacije
Avtonomnost	ali so enote resnično porazdeljene in nimajo centralnega nadzora
Zanesljivost	redundanca, večkratno preverjanje in triangulacija rezultatov
Omejeni viri	upoštevati je potrebno zmogljivosti sistema

2.2 Večagentni sistemi

Za reševanje večjih problemov je smiselno združiti agente v večagentni sistem. Taki sistemi imajo pred posameznimi agenti veliko prednosti. Zaradi številčnosti lahko rešujejo obsežnejše probleme, za katere bi potrebovali zelo dragega agenta, če bi jih reševal sam. Prav tako lahko za rešitev določenega problema porabijo veliko manj časa, kot bi ga potreboval en sam agent.

Za sistem agentov potrebujemo zasnovo okolja, v katerem lahko agenti uspešno sodelujejo. Zasnova okolja mora vsebovati protokole za komunikacijo in protokole za interakcijo med agenti. Komunikacijski protokoli omogočajo izmenjavo in razumevanje sporočil, interakcijski protokoli pa omogočajo agentom, da se med seboj pogovarjajo. Interakcijski protokoli so strukturirane izmenjave sporočil (pogovori).

Primer **komunikacijskega protokola**, ki ga lahko omogoča okolje, je na primer:

- agenta predlagata naslednjo akcijo
- agenta sprejmeta naslednjo akcijo
- agenta zavrneta naslednjo akcijo
- agenta odvzameta predlog za naslednjo akcijo
- agenta se ne strinjata z naslednjo akcijo
- agenta predlagata nadomestno naslednjo akcijo

Glede na zgornji komunikacijski protokol bi lahko bil primer interakcije med dvema agentoma:

- Agent 1 predlaga akcijo Agentu 2
- Agent 2 oceni predlog in
  - sprejme in obvesti Agenta 1
  - predlaga nadomestno akcijo in obvesti Agenta 1
  - se ne strinja s predlogom in obvesti Agenta 1
  - zavrne predlog Agenta 1

Možnost sodelovanja z ostalimi agenti prisili razvijalca, da drugače razmišlja o problemu in razvoju agenta. Razvijalec se tako vpraša kaj agent zna in kako lahko ostali agenti dostopajo do tega znanja. Tako je agent razvit veliko bolj deklarativno[1], namesto tipičnega proceduralnega razvoja.

2.2.1 Karakteristike večagentnega okolja

Več agentno okolje mora agentom omogočati infrastrukturo, ki specificira komunikacijske in interakcijske protokole. Taka okolja so ponavadi odprta in

[1] Agenti so razviti tako, da jim povemo kaj želimo z njimi narediti in ne kako želimo to narediti. Namesto navodil uporabimo dejstva in pravila. Primer deklarativnega programskega jezika je recimo Prolog, kjer mora prevajalnik sam ugotoviti kako narediti, kar želimo narediti.

Tabela 2.2: Karakteristike okolja, delno povzeto po [19].

Komunikacijska infrastruktura	deljen pomnilnik, sporočila, točka-v-točko
Protokol za sporočila	http, CORBA, protobuffer
Razreševanje konfliktov	sinhronizacija, več-nitnost
Varnost	avtentikacija, časovne oznake, omejenost
Podpora operacijam	arhiviranje, redundanca

nimajo centraliziranega snovalca oziroma razvijalca. Okolje vsebuje agente, ki so avtonomni in porazdeljeni in jim je omogočeno delovanje za svoje interese ter imajo možnost, da za dosego ciljev med seboj sodelujejo.

Pri zasnovi takega okolja moramo biti pozorni na nekatere ključne karakteristike naštete v Tabeli 2.2. Če okolje spoštuje karakteristike, je omogočeno sobivanje agentov, ki imajo zmožnost komunikacije in interakcije, ter so zmožni sodelovanja za skupno reševanje problemov.

Agenti, ki v okolju delujejo, morajo biti z okoljem povezani. Pomembne povezave med agentom in okoljem so naštete v Tabeli 2.3. Agent mora imeti o okolju neko znanje in zmožnost predvidevanja okolja. Okolje pa mora beležiti agentovo zgodovino, ter spremljati njegovo trenutno stanje.

2.2.2 Komunikacija med agenti

Agent je aktiven objekt, ki ima zmožnosti razpoznavanja, razumevanja in lahko vpliva na okolje. Ima neko reprezentivno znanje in mehanizem za sklepanje iz svojega znanja. Prav tako ima zmožnost komuniciranja. Ta je del razpoznavanja (zmožen je prebrati prejeto sporočilo) in tudi del akcije (zna poslati sporočila). V simuliranem okolju (na računalniku) so to morda edine razpoznavne možnosti in edini možen vpliv v okolju.

Tabela 2.3: Povezava agenta z okoljem, delno povzeto po [19].

Znanje	Koliko okolja agent pozna in ali lahko o okolju poizveduje?
Predvidljivost	Koliko okolja lahko agent predvidi?
Nadzor	Ali lahko agent spreminja okolje?
Zgodovina	Ali so prihodnja stanja odvisna od celotne zgodovine ali le od trenutnega stanja?
Namen	Kakšen je agentov namen v okolju in ali so tam še ostali agenti?
Realno-časovnost	Ali se okolje spreminja med agentovo interakcijo?

Koordinacija

Komunikacija omogoča agentom, da koordinirajo svoje akcije in obnašanje, kar omogoča bolj konsistentne in organizirane sisteme.

Stopnja koordinacije določa kako dobro se agenti izmikajo nezaželjenim akcijam. Koordinacija omogoča tudi zmanjšanje boja za vire, ki so agentom na voljo in izogibanje t.i. mrtvi in živi zanki (ang. "deadlock"[2] in "livelock"[3]) problema, ter ohranjanje varnosti v sistemu. Sodelovanje je koordinacija med ne-tekmovalnimi agenti, medtem ko je pogajanje koordinacija med agenti, ki med seboj tekmujejo. Za uspešno sodelovanje, mora vsak agent poznati model ostalih agentov in razviti model za prihodnje interakcije - tako se razvije socialnost agentov.

Skladnost nam pove kako dobro se sistem obnaša kot enota. Problem za večagentne sisteme je kako ohranjati globalno skladnost brez eksplicitnega

[2]Je problem, kjer dve akciji oz. agenta čakata en na drugega in posledično se nič ne zgodi, sistem se ustavi.

[3]Je možna posledica reševanja iz problema mrtve zanke, če več agentov hkrati izvede določeno akcijo. V tem primeru se sistem ne ustavi, agenti so aktivni vendar ne napredujejo.

globalnega nadzora. V tem primeru morajo agenti sami določiti cilje, ki si jih delijo z drugimi agenti, ugotoviti katere naloge so jim skupne, izogibati se morajo konfliktom ter se morajo zavedati svojega repozitorija znanja in dejstev. Zato pomaga, če je med agenti implementirana vsaj nekakšna organizacija in dodeljevanje nalog.

Dimenzije pomena

Formalno komunikacijo določa sintaksa (kako so simboli strukturirani), semantika (kaj simboli pomenijo) in pragmatika (kako so simboli interpretirani). Pomen je kombinacija semantike in pragmatike.

Na pomen sporočila vpliva več faktorjev. Med pomembnejši so *številnost* (zasebna sporočila imajo drugačen pomen kot javna sporočila), *kontekst* (sporočilo mora biti inerpretirano glede na mentalno stanje agenta, trenutno stanje okolja, zgodovino okolja, kako je prispelo) in *identiteta* (kdo je poslal sporočilo in kakšno vlogo ima v sistemu).

Tipi sporočil

Agenti morajo biti zmožni komunicirati tudi z najmanj zmožnimi agenti, saj drugače slednji izgubijo interes. Vloga v pogovoru je lahko aktivna ali pasivna ali oboje. Dve osnovni vrsti sporočil sta poizvedba in trditev. Vsak agent (tako aktiven kot pasiven) mora biti zmožen sprejemati informacije.

Za pasivno obnašanje mora biti agent zmožen sprejeti poizvedbo in odgovoriti s trditvijo. Za aktivno obnašanje pa mora biti agent zmožen poizvedovati in postavljati trditve. S tem lahko celo pridobi nadzor nad pasivnim agentom. Večinoma so agenti tako pasivni in aktivni, torej so zmožni poizvedovanja, sprejemanja trditev in odgovarjanja s trditvami.

Nivo komunikacije

Komunikacijski protokoli so ponavadi specificirani na večih nivojih. Najnižji nivo opisuje medpovezljivost, srednji nivo opisuje format oz. sintakso inforamcije, ki se pošilja. Na višjem nivoju pa se specificira pomen in semantika informacije. Semantika se nanaša tako na tip sporočila kot tudi na vsebino sporočila.

2.2.3 Interakcijski protokoli

Interakcijski protokoli narekujejo izmenjavo večih sporočil med agenti, narekujejo torej pogovor. Če imajo agenti konfliktne cilje ali so zainteresirani samo za svoje cilje uporabimo protokole, ki maksimizirajo izkupiček (ang. "utilities") za agente. Kjer imajo agenti skupne cilje ali problem, kot v porazdeljenih problemih, je cilj protokola vzdrževanje globalne skladnosti sistema agentov brez kršitve avtonomije (brez globalnega nadzora). Pomembni aspekti so kako določiti skupne cilje, skupne naloge, se izogibati nepotrebnim konfliktom in vzdrževanje repozitorija znanja in dejstev [19].

2.2.4 Koordinacijski protokoli

V okolju z omejenimi viri morajo agenti aktivnosti med seboj koordinirati, da lahko zadovoljijo svoje ali skupne cilje. Koordinacija je pomembna saj obstajajo odvisnosti akcij med agenti in ni potrebe po zadovoljevanju globalnih omejitev. Prav tako posamezen agent nima kompetenc, virov ali informacij za dosego cilja celotnega sistema. Primer koordinacije je recimo posredovanje pravočasnih informacij ostalim agentom, skrb za sinhronizacijo akcij agentov in izgobianje redundantnemu reševanju problemov.

Za koordinacijo je potrebno poskrbeti za porazdelitev nadzora in tudi podat-

kov. Porazdelitev nadzora pomeni, da imajo agenti avtonomnost pri izbiri naslednjega cilja in generiranju akcij. Problem porazdeljenega nadzora in podatkov pa je, da je znanje celotnega sistema porazdeljeno čez celoten sistem in ima posamezen agent samo delno perspektivo. Zaradi tega je povišana stopnja negotovosti akcije vsakega agenta in posledično je težje vzdrževati skladno globalno obnašanje.

Ključne agentne strukture so obveznosti (ang. "commitments") in konvencija. Obveznosti prisilijo agenta, da se zavzema za določene akcije, konvencija pa omogoča upravljanje z obveznostmi glede na spreminjajoče okoliščine.

Obveznosti in konvencije so temelji koordinacije: Obveznosti omogočajo potrebno strukturo za predvidljivo interakcijo in socialne konvencije omogočajo potrebno stopnjo skupne podpore [19].

2.2.5 Protokoli za sodelovanje

Večina protokolov za sodelovanje se osredotoča na razgraditev in porazdelitev nalog med agente. S pristopom deli in vladaj je mogoče znižati kompleksnost obsežne naloge. Manjše podnaloge zahtevajo manj sposobne agente in manj obremenijo vire, ki jih imajo agenti na voljo, kar omogoča preprostejše agente.

Pri porazdelitvi nalog je potrebno izogibanje preobremenjevanju ključnih virov in da so naloge enakomerno porazdeljene med agente z zadostnimi zmožnostmi. V nekaterih primerih lahko agenti s širšim pogledom razdelijo naloge ostalim agentom. Prav tako je potrebno paziti, da se prekrivajoče naloge razdelijo na način, ki ohranja skladnost sistema. Močno odvisne naloge lahko razdelimo med agente, ki so v prostorski ali pomenski bližini in tako zmanjšamo potrebno količino komunikacije in sihronizacije.

Poglavje 3

Preiskovanje prostora z večagentnim sistemom

Področje preiskovanja prostora z večimi agenti (in v resničnem svetu z večimi roboti) je raziskovalno zelo aktivno in gre z roko v roki s problemom mapiranja prostora ter lokalizacijo agenta v prostoru (če agent rešuje oba problema hkrati, je ta problem poznan kot *SLAM* (sočasna lokalizacija in mapiranje prostora, ang. "Simultaneous localization and mapping"). V resničnem svetu ima preiskovanje prostora z večimi roboti veliko možnosti uporabe, med katerimi so tudi reševanje iz odročnih in težje dostopnih krajev, preiskovanje podvodnih in nadvodnih jam, preiskovanje ogromnih nepoznanih površin (recimo planeti), iskanje pomembnih tarč in še mnogo drugih.

3.1 Algoritmi in strategije preiskovanja

Definicija problema preiskovanja prostora povzeta po [22] je "Glede na trenutno poznavanje prostora, kam se premakniti, da pridobimo čim več nove informacije?" Cilj preiskovalnih algoritmov je torej pridobiti največ

informacije o nepoznanem okolju v najkrajšem možnem času.

Še danes se za preiskovanje prostora v osnovi uporablja algoritem, ki ga je razvil avtor zgornje definicije. Za implementacijo algoritmov se prostor razdeli (diskretizira) na celice in se prekrije z navidezno mrežo. Glavna ideja algoritma je premik agentov na obmejno celico, ki je tista celica, ki je na robu že preiskanega in meji na nepreiskani del prostora.

Vendar pa ima algoritem tudi nekaj pomankljivosti. [20] predstavi učinkovit algoritem za določanje obmejne celice ampak se agenti odločajo individualno, kar pomeni da med njimi ni eksplicitne koordinacije in posledično lahko več agentov raziskuje isti prostor ter se med seboj morda celo motijo. Zato [2] predstavijo učinkovito koordinacijo za agente. V nadaljnih raziskavah [3] se osredotočijo na minimizacijo potrebnega časa za raziskovanje različnih regij okolja, kar privede do algoritmov za razporejanje nalog in algoritmov, ki so zasnovani na statističnih metodah. Slabost takih algoritmov pa so dragi senzorji in potrebna računska zmogljivost agentov, kar vpliva tudi na skalabilnost celotnega sistema.

Podoben pristop z dodeljevanjem obmejnih celic so razvili tudi [1], to pa so poskusili narediti brez potrebe po komunikaciji med agenti, kar poenostavi algoritme in omogoča uporabo manj zmogljivih in cenejših agentov. Avtorji predpostavljajo, da so agenti zmožni rešiti problem SLAM in se osredotočijo na izbiro podnalog za vsakega agenta, dodeljevanje nalog in frekvenco preračunavanja dodeljevanja nalog. Ker ne vemo kaj se skriva za obmejno celico je v vsakem koraku potrebna optimizacija, zato algoritem uporabi optimizacijske kriterije, da ugotovi uporabnost dodeljene celice. Prav tako morajo kriteriji potrditi, da se je enemu agentu dodelila samo ena obmejna celica.

Vendar pa je algoritem z obmejnimi celicami le en način, kako se lotiti problema preiskovanja, zato so raziskali tudi drugačne pristope.

Algoritmi za preiskovanje prostora se delijo na tri večja področja

I naključni preiskovalni algoritmi, npr. [4]

II algoritmi, ki temeljijo na planiranju, npr. [3]

III algoritmi, ki temeljijo na optimizaciji roja delcev (PSO) in posnemajo naravne pojave, npr. [13]

Lastnosti preiskovalnega algoritma
Dober preiskovalni algoritem mora imeti dve lastnosti, zaključenost in učinkovitost. Agent mora torej pokriti večino prostora, ki ga raziskuje, učinkovitost pa narekuje, da doseže zaključek s čim manj truda, v čim krajšem času z nizko porabo energije.

Zakaj uporabljati sodelujoče agente?
Sodelujoči agenti imajo potencial, da dosežejo nalogo hitreje, kot en sam agent. Več agentov lahko preiskuje prostor bolj učinkovito, če si izmenjujejo določeno informacijo, kot so pozicije in že preiskana področja, zato se izognemo ponovnemu preiskovanju že preiskanega.

3.2 Naključni preiskovalni algoritmi

Ker algoritmi, ki temeljijo na planiranju in računanju uporabnosti celic potrebujejo natančno lokalizacijo in posledično drage agente so te zahteve poskusili zmanjšati tako, da so agenti usklajeni v formacijah in vidijo en drugega. Vendar to ne izkoristi paralelizma v porazdeljenih sistemih v celoti. Zato so prednosti naključnih algoritmov [10] v nižji ceni implementacije in njihove sposobnosti niso odvisne od senzorjev in zmožnosti lokalizacije (v resničnem svetu na robotih).

Kot primer takega algoritma, [4] predstavijo računsko nezahteven naključen algoritem za preiskovanje prostora, ki ne zahteve lokalizacije. Agenti sledijo

določenim pravilom obnašanja:

1. Izogibaj se oviram in ostalim robotom.
2. V prostoru išči tarčo (vir svetlobe) in opozori sosednje robote v primeru, da jo zaznaš.
3. Odgovarjaj sporočilom sosednjih robotov.
4. Raziskuj okolje (naključno).

[4] uspešno dokažejo, da naključen algoritem dosega dobre rezultate, pri tem pa porabi malo računskih virov in se ne obremenjuje z zanesljivim dostavljanjem sporočil.

3.3 Algoritmi s planiranjem in dodeljevanjem nalog

Tako kot drugi so tudi avtorji [2] poskusili čim bolj zmanjšati čas potreben za preiskovanje prostora. Glavna ideja avtorjev je izbira ciljnih točk za posamezne agente, tako da lahko sočasno raziskujejo različne dele prostora. Predstavijo algoritem za koordinacijo, ki upošteva ceno in uporabnost ciljne celice. Uporabnost celice je velikost prostora, ki ga lahko agent odkrije s senzorji iz ciljne točke. Ko se ciljna točka dodeli agentu, se njena uporabnost za ostale agente v sistemu zmanjša. Tako si lahko skupina agentov razdeli različne točke. Eden izmed ključnih problemov je koordinacija agentov, saj brez tega lahko več agentov preiskuje isti prostor. Zato je potrebno izbrati različne akcije za različne agente, tako da sočasno preiskujejo različen prostor. V članku [21] je opisano kako več agentov učinkovito združi mapo pri preiskovanju prostora, ki si jo agenti med seboj delijo in tako vedo, kateri deli prostora so že bili preiskani.

V [21] se agenti pomikajo proti najbližji obmejni celici, ki je najbližje neznani okolici glede na trenutno mapo. Vendar pa ne uporablja koordinacijske

komponente, ki bi izbirala različne celice za različne agente. Pristop v [2] pa reši ravno to. Dodatna izboljšava, ki jo predlagajo avtorji je računanje razdalje glede na trenutno mapo, namesto "zračne" razdalje, ki je manj natančna.

Bistveno je, da agenti vedo kateri deli prostora so že bili raziskani. Da lahko roboti uskladijo svoje akcije in planirajo poti, morajo zgraditi globalno mapo. V članku uporabijo mrežo zasedenosti (ang."occupancy-grid") mape za predstavo okolja. Prav tako beležijo kateri deli so že bili raziskani, da lahko identificirajo nove ciljne celice. Predpostavijo, da je novo odkrito področje veliko toliko kolikor lahko zaznajo agentovi senzorji, ko pride na cilj in to informacijo uporabljajo za izbiro pozicij ostalih agentov. Predpostavljajo tudi, da agenti poznajo relativne pozicije med preiskovanjem.

3.4 Optimizacija roja delcev

Roji agentov so precej nov pristop k koordinaciji velikega števila relativno enostavnih fizičnih robotov ali agentov, ki so avtonomni, brez centralnega nadzora in so zmožni lokalne komunikacije, ki posnemajo pojave v naravi (čebele, ptice, mravlje, volkovi, ...).

Prednosti takih algoritmov so možnost paralelizma, robustnost (zaradi številčnosti), skalabilnost zaradi preprostosti in porazdeljenosti, heterogenost (imamo lahko različno zmožne agente), prilagodljivost in zmožnost reševanja zelo kompleksnih nalog ter majhna cena zaradi enostavnosti posameznih agentov [15].

Način komunikacije pri PSO se deli na implicitno in eksplicitno komunikacijo. Implicitna (imenuje se tudi stigmetrija) je način komuniciranja skozi okolje (uporaba virtualnih feromonov, ki posnema naravno komunikacijo, kot na primer pri mravljah, ki kot sledove v okolju puščajo feromone), eksplicitna

pa je direktna komunikacija med agenti. Dva načina je primerjal McPartland [14]. Oba sta se izkazala za uspešen način komunikacije, v diplomskem delu pa se bomo osredotočili na eksplicitno komunikacijo.

V osnovi je optimizacija roja delcev način optimizacije nelinearnih zveznih funkcij s široko uporabnostjo. Predstavila sta ga [9]. Osnovna ideja je, da uporabimo roj delcev, ki išče rešitev v ponavadi več-dimenzionalnem hiperprostoru problema, s pomočjo funkcije uspešnosti (ang. "fitness function"). En delec si lahko predstavljamo kot potencialno rešitev problema. Cilj je, da vsi delci odkrijejo globalno najboljšo rešitev. Prednosti so predvsem v enostavnosti osnovnega algoritma in nizki časovni kompleksnosti, saj algoritem zahteva le zelo osnovne operacije. Je soroden genetskim algoritmom vendar si je zmožen zapomniti zgodovino delcev, medtem ko se pri genetskih algoritmih identiteta lahko ohranja, ali pa ne, le za najbolj elitne osebke. Največja podobnost je v tem, da rešujeta sorodne probleme in oba pristopa se inicializirata s populacijo naključnih rešitev.

Vsak delec v roju si beleži svojo trenutno najboljšo rešitev, ki jo poimenujemo "**pbest**". Beleži si tudi vrednost "**gbest**", ki je najboljša rešitev celotnega roja, kar je hkrati edina informacija, ki si jo mora roj delcev sporočati. V vsaki iteraciji nato delec pospešuje (se premika proti) svoji najboljši in globalni najboljši rešitvi. Algoritem se izvaja, dokler ne doseže vnaprej določenega kriterija. Edina težava algoritma je možnost, da se zatakne v lokalnih optimumih[1]. Za ta problem so kasneje raziskovalci razvili več rešitev, eno pa predlagata že [9].

Ker je PSO algoritem že v osnovi porazdeljen, brez centralnega nadzora, nima visoke računske zahtevnosti (vsak delec je lahko zelo enostaven) in ima zelo učinkovito komunikacijo (v osnovi agenti potrebujejo sporočati

[1]Problem lokalnih optimumov je problem v optimizaciji, kjer algoritem najde rešitev, ki je optimalna vendar le v bližnji okolici. Cilj algoritma pa je poiskati globalno rešitev, ki je najboljša rešitev med vsemi možnim.

le en podatek), smo se odločili da algoritme na osnovi PSO raziščemo bolj podrobno in enega izmed njih tudi implementiramo, ter ga primerjamo z lastnim pseudo-naključnim algoritmom.

3.4.1 Osnovni algoritem PSO

Osnovni algoritem inicializira delce v vseh dimenzijah prostora problema, kjer vsak delec predstavlja potencialno rešitev problema. Delci se premikajo skozi prostor in iščejo globalni optimum (najboljšo rešitev). Oznake uporabljene v zapisu Algoritma 1 in 2 so zbrane v Tabeli 3.1.

Ob inicializaciji, ki je navedena v Algoritmu 1, se delci naključo generirajo skozi celoten prostor. Kot najboljšo pozicijo jim nastavimo kar njihovo trenutno pozicijo. Ko imajo vsi delci določeno najboljšo pozicijo, določimo še globalno najboljšo pozicijo. Nadaljevanje je opisano z Algoritmom 2, kjer v vsaki iteraciji najprej posodobimo lokalne in globalne najboljše vrednosti, nato pa posodobimo še hitrost in pozicijo delcev. Algoritem se ponavlja dokler ne izpolnimo pogoja (ko algoritem skonvergira) ali dosežemo vnaprej določenega števila iteracij.

3.4.2 Uporaba PSO za iskanje ciljev v prostoru

Kot primer uporabe osnovne optimizacije roja delcev za iskanje ciljev v prostoru so [13] uporabili algoritem Porazdeljena optimizacija roja delcev [12] (PORD, ang. “distributed particle swarm optimisation” - dPSO). Ta algoritem ima veliko prednosti, saj ne potrebuje zahtevnih kalkulacij, kar ustreza njihovemu namenu, da za preizkušanje uporabijo zelo majhne robote (ki so odvisni od avtonomije baterije in imajo nezmogljiv procesor). Poleg tega omogoča ogromno skalabilnost sistema, saj se število informacije za komunikacijo z dodajanjem agentov ne spreminja veliko.

Tabela 3.1: Oznake predstavljene v Algoritmu 1 in 2.

$\mathbf{n}$	Indeks delca v roju.
$\mathbf{d}$	Dimenzija problema. Vsak delec ima hitrost in pozicijo v vsaki dimenziji.
$\mathbf{x_{n,d}[t]}$	Pozicija delca n v dimenziji d, v času t.
$\mathbf{v_{n,d}[t]}$	Hitrost delca n v dimenziji d, v času t.
$\mathbf{c_1}$	Konstanta za kognitivno komponento.
$\mathbf{c_2}$	Konstanta za socialno komponento.
Rand	Naključna vrednost med 0 in 1.
$\mathbf{pbest_{n,d}[t]}$	Pozicija v dimenziji d, ki je najbolj ustrezna od vseh obiskanih s strani delca n v dimenziji d in času t.
$\mathbf{gbest_d[t]}$	Pozicija v dimenziji d, ki je najbolj ustrezna od vseh obiskanih v dimenziji d in času t.

Vsak agent je delec v PSO. Vsak agent meri in posodablja svojo pozicijo, svojo hitrost in posodablja svojo najboljšo meritev ("pbest") in najboljšo lokacijo (če je to potrebno) ter jo posreduje ostalim, če najde globalno najboljšo meritev oz. pozicijo.

Za algoritem si zamislijo nekaj lastnosti:

- porazdeljen med mnogo robotov
- računsko preprost
- skalabilen iz 10 na 100 in tudi tisoče agentov

Vsak agent si beleži lokalno najboljšo vrednost (*"pbest"*) funkcije, ki oceni agentovo rešitev problema (izmerjena najvišja svetlost), hkrati pa poskuša odkriti globalno najboljšo vrednost (*"gbest"*). V primeru, da najde novo najboljšo globalno vrednost pošlje to informacijo ostalim. To je edina informacija, ki jo roboti potrebujejo, da izračunajo novo hitrost in pozicijo.

Algoritem 1 Inicializacija PSO, povzeto po [17]

for all delec n v roju S **do**
  for all dimenzija d v prostoru D **do**
    //inicializiraj lokacije in hitrosti delcev
    $x_{n,d}[t] = \textbf{Rand}\,(x_{min}[t], x_{max}[t])$
    $v_{n,d}[t] = \textbf{Rand}\,(-v_{max}[t], v_{max}[t])$
  end for
  //inicializiraj najboljšo pozicijo delca
  $pbest_n[t] = x_n[t]$
  if $f(pbest_n[t]) < f(gbest_n[t])$ **then**
    //posodobi gbest vrednost
    $gbest[t] = pbest_n[t]$
  end if
end for

Algoritem je uspešno našel vir svetlobe, ki je bil uporabljen kot cilj. Avtorji so pokazali, da je optimizacija roja delcev primerna za uporabo v takih nalogah.

Pomankljivosti tega algoritma pa so v tem, da ni optimiziran za paralelizacijo v primeru uporabe kompleksnejših iskalnih algoritmov (to je storil [5]) in pa možnost zatikanja v lokalnih optimumih. Cilj algoritma je namreč zminimizirati potrebno informacijo, ki jo je potrebno pošiljati med agenti, kar omogoča boljšo skalabilnost sistema (na tisoče preprostih agentov). Prav tako avtorji niso upoštevali morebitnega izogibanja oviram, v primeru ovire je imel robot vnaprej sprogramirano obnašanje.

Algoritem 2 Optimizacija roja delcev, povzeto po [17]

repeat
  for all delec n v roju S **do**
    //posodobi najboljšo lokalno vrednost
    if $f(x_n[t]) < f(pbest_n[t])$ **then**
      $pbest_n[t] = x_n[t]$
    end if
    if $f(pbest_n[t]) < f(gbest_n[t])$ **then**
      //posodobi gbest vrednost
      $gbest[t] = pbest_n[t]$
    end if
  end for
  //posodobi hitrost in pozicijo delcev
  for all delec n v roju S **do**
    for all dimenzija d v prostoru D **do**
      $v_{n,d}[t+1] = v_{n,d}[t] + c_1 * \mathbf{Rand}(0,1) * [pbest_{n,d}[t] - x_{n,d}[t]] + c_2 * \mathbf{Rand}(0,1) * [gbest_d[t] - x_{n,d}[t]]$
      $x_{n,d}[t+1] = x_{n,d}[t] + v_{n,d}[t+1]$
    end for
  end for
  //povečaj število iteracij
  št.iteracij $+= 1$
until št. iteracij $<$ MAX_ŠT_ITERACIJ

3.5 RPSO, DPSO in RDPSO

3.5.1 RPSO

Za uporabo optimizacije roja delcev v večagentnih sistemih je bila kasneje predstavljena nadgradnja, saj se morajo v realnem svetu agenti izogibati oviram, ki jih zaznajo s svojimi senzorji. Za ta namen so razvili algoritem Robotska optimizacija roja delcev (RPSO, ang. "Robot particle swarm optimisation") [7].

RPSO dopolni PSO tako, da mu doda zmožnost izogibanja oviram, tako da pri izračunu nove hitrosti delca upošteva še komponento za izogibanje oviram. Predpostavljeno je, da je agent zmožen zaznati oviro v določenem polmeru r_s. Za funkcijo zaznavanja ovire je predstavljena funkcija $g(x_{n,d}[t])$, ki je pozitivna. Odvisna je od razdalje agenta do ovire. Tako se hitrost delca posodobi po (3.1), razlaga uporabljenih oznak pa je predstavljena v Tabeli 3.2.

$$\begin{aligned} v_n\,[t+1] = {} & w v_{n,d}\,[t] + c_1 \mathbf{Rand}(0,1) * (g\vec{best}_d\,[t] - x_{n,d}[t]) \\ & + c_2 \mathbf{Rand}(0,1)(\vec{x_{n,d}}[t] - x_{n,d}[t]) + c_3 \mathbf{Rand}(0,1)(\vec{x_n^{gbest}}[t] - x_{n,d}[t]) \end{aligned} \quad (3.1)$$

Tabela 3.2: Oznake predstavljene v (3.1).

$\mathbf{w}$	Koeficient inercije
$\mathbf{c_3}$	Konstanta za izogibanje oviram.
$\vec{\mathbf{x}_n^{gbest}}[t]$	Pozicija delca $\mathbf{n}$, ki optimizira funkcijo $g(x_{n,d}\,[t])$.

Če v okolju ni ovir, faktor $\mathbf{c_3}$ postavimo na 0. Tako primerna izbira faktorjev $\mathbf{c_1}$, $\mathbf{c_2}$, $\mathbf{c_3}$ agentom narekuje ravnotežje med doseganjem glavnega cilja in

izogibanjem oviram. Če velja

$$\mathbf{c_3} << min(\mathbf{c_1}, \mathbf{c_2})$$

so agenti bolj naklonjeni doseganju cilja in niso tako pozorni na ovire. Če velja

$$\mathbf{c_3} >> min(\mathbf{c_1}, \mathbf{c_2})$$

pa agenti lahko potrebujejo več časa za dosego ciljev, vendar se bolj pozorno izogibajo oviram.

Kljub nadgradnji osnovnih algoritmov in komponenti za izogibanje oviram RPSO še vedno ne odpravi možnosti zatikanja v lokalnih optimumih. Najnovejši algoritem, ki uporablja optimizacijo roja delcev v namen iskanja in reševanja (ang. S&R - "search and rescue") ter preiskovanja prostorov pa je Robotska Darwinova optimizacija roja delcev (ang. "Robotic Darwinian Particle Swarm Optimisation" - **RDPSO**) [6], ki odpravi še problem zatikanja v lokalnih optimumih.

3.5.2 DPSO

Za rešitev problema z lokalnimi optimumi se avtorji [7] poslužijo algoritma DPSO (ang. "Darwinian Particle Swarm Optimisation"), ki temelji na naravani selekciji [18]. Za uporabo algoritma na večagentnih sistemih so razvili algoritem RDPSO (Algoritem 4).

DPSO (Algoritem 3) posnema naravno selekcijo, tako da roj delcev razdeli na podskupine, ki jih lahko zaradi napredovanja nagradimo z dodatnimi delci (*podskupino nagradimo*) ali pa jim jih zaradi slabega napredovanja odvzamemo (*podskupino kaznujemo*). Koncept se imenuje izključevanje in vključevanje v skupine. Tako posnemamo preživetje najmočnejših delcev, ki pripomorejo k uspehu posamezne podskupine. S tem, ko podskupine med seboj tekmujejo,

se izognejo zatikanju v lokalne optimume, saj v primeru, da se ena podskupina zatakne, ostale še vedno napredujejo k boljši rešitvi. Podskupini, ki se zatakne, se kmalu izteče življenjska doba in ostane brez delcev, medtem ko ostale podskupine dobijo dodatne delce.

Algoritem 3 Darwinova Optimizacija roja delcev, povzeto po [6]

repeat
  for all podskupina v celotnem roju **do**
    razvij_podskupino()
    dodeli podskupini novega agenta //če ustreza zahtevam
    razpusti slabe podskupine
  end for
until zaustavitveni kriterij oz. konvergenca

Function razvij_podksupino()
for all delec n v podksupini **do**
  oceni ustreznost delca n
  posodobi $pbest_n[t]$ in $gbest_n[t]$
  posodobi $v_n[t+1]$ in $x_n[t+1]$

  if podskupina napreduje **then**
    nagradi podskupino: dodaj delec in podaljšaj življenjsko dobo
  end if
  if podskupina ne napreduje **then**
    kaznuj podskupino: po potrebi izbriši delec in znižaj življenjsko dobo
  end if
end for

3.5.3 RDPSO

S pomočjo algoritma DPSO so odpravljene pomankljivosti zatikanja v lokalnih optimumih, algoritem RPSO pa doda možnost izogibanja oviram. Oba algoritma sta združena v novi algoritem RDPSO, ki je prilagojen za večagentni sistem v realnem svetu.

Tako kot v originalnem algoritmu PSO je tudi v algoritmu RDPSO N delcev, ki potujejo skozi prostor in iščejo globalno optimalno rešitev. Za posodobitev hitrosti se uporablja (3.1). Delci pa se razdelijo v podskupine, ki so lahko za uspešnost nagrajene z dodatnimi delci ali pa za neuspešnost kaznovane z odvzemom delcev. Da obstaja skupina delcev, mora biti število delcev v skupini večje od $\mathbf{n_{min}}$. Če ta pogoj ni izpolnjen, se podskupina razpusti in njeni člani so dodani v skupino izločenih agentov. Agenti v skupini izločenih se kasneje ponovno dodelijo skupinam, ki so uspešne. V uspešno skupino se dodeli najboljši izmed agentov, ki je v skupini izločenih.

Če podskupina naraste, ima v vsakem koraku možnost, da ustvari novo podskupino z $\mathbf{n_i}$[2] najboljšimi agenti v skupini izločenih. Ta možnost je omejena z verjetnostjo $\mathbf{p_{sp}}$, ki je definirana kot (3.2) (oznake v Tabeli 3.3) in s pogojem, da v podskupini nikoli ni bil izbrisan delec. To prepreči ustvarjanje novih podskupin, če jih obstaja že veliko.

$$p_{sp} = \frac{r_{sp}}{NS} \tag{3.2}$$

Tabela 3.3: Oznake predstavljene v (3.2).

NS	število podskupin
$\mathbf{r_{sp}}$	naključna vrednost med 0 in 1

[2] $\mathbf{n_i}$ določa začetno velikost poskupine, $\mathbf{n_{min}}$ določa minimalno velikost podskupine in $\mathbf{n_{max}}$ določa največjo velikost podskupine

Za omejevanje razvoja podskupine je uporabljen števec $\mathbf{SC_s}$, ki šteje kolikokrat se podskupina s razvije brez, da najde boljšo rešitev. Če podskupina preseže konstantno $\mathbf{SC_{max}}$ se iz podskupine izključi najslabšega agenta. Kater agent je najslabši, se določi s primerjavo njegove rešitve z ostalimi agenti v isti podskupini. $\mathbf{SC_s}$ se potem ponastavi po (3.3) (oznake v Tabeli 3.4).

$$SC_s = SC_{max}\left[1 - \frac{1}{N_s^{kill} + 1}\right] \tag{3.3}$$

Tabela 3.4: Oznake predstavljene v (3.3).

$\mathbf{SC_s}$	števec razvoja podskupine s
$\mathbf{N_s^{kill}}$	število delcev izbrisanih iz podskupine s, v času, ko ni bilo napredka
$\mathbf{SC_{max}}$	maksimalno število korakov brez, da se izboljša ustreznost skupine

Agenti, ki pripadajo skupini izločenih ne iščejo globalnega optimuma, pač pa naključno potujejo po prostoru, vendar se vedno zavedajo svoje lokalne rešitve ("pbest") in globalne rešitve skupine izločenih ("gbest").

Osnutek algoritma RDPSO je predstavljen v Algoritmu 4, podrobno je opisan v Algoritmu 5, oznake pa so predstavljene v Tabeli 3.5.

Tabela 3.5: Oznake predstavljene v Algoritmu 5.

$\mathbf{h_n[t]}$	evklidska razdalja, ki jo mora prepotovati agent n
$\chi_i[t]$	predstavlja vektor najboljših rešitev za kognitivno in socialno komponento, ter komponento za izogibanje oviram, $i = 1, 2, 3$
$\mathbf{N_s}$	število podskupin
$\mathbf{N_s^{max}}$	maksimalno število podskupin

Algoritem 4 Osnutek RDPSO, povzeto po [7]

repeat
  for all podskupina v celotnem roju **do**
    razvij_podskupino()
    dovoli podskupini, da naredi novo podskupino iz n_i najboljših agentov v skupini izključenih
    neuspešne skupine razpusti in dodaj agente v skupino izključenih
  end for
until zaustavitveni kriterij oz. konvergenca

Function razvij_podksupino()
for all agent n v podksupini s **do**
  posodobi funkcijo uspešnosti agenta $f(x_n[t])$
  posodobi $pbest_n[t]$
  premakni agenta (posodobi pozicijo in hitrost)

  if podskupina napreduje **then**
    nagradi podskupino s: dodaj najboljšega agenta iz skupine izločenih
  end if
  if podskupina ne napreduje **then**
    kaznuj podskupino s: po potrebi izloči najslabšega agenta v skupino izločenih
  end if
end for

Algoritem 5 Algoritem RDPSO, povzeto po [6]

inicializiraj pozicijo $x_n[0]$ in $swarmID$
repeat
  if swarmID $\neq 0$ **then** //preveri, če agent ni v skupini izključenih
    oceni rešitev agenta $h_n[t]$
    if $h(x_n[t]) > h_{best}$ **then** //če se je agent izboljšal
      $h_{best} = h(x_n[t])$
      $\chi_1[t] = x_n[t]$ //posodobi najboljšo kognitivno komponento
    end if
    Izmenjaj informacije s preostalimi agenti skupine. Posreduj svojo rešitev $h_n[t]$ in trenutno pozicijo $x_n[t]$. Zgradi vektor $H[t]$, ki vključuje rešitve vseh agentov znotraj podskupine $swarmID$.
    if $maxH[t] > H_{best}$ **then** //preveri če se je podksupina izboljšala
      $H_{best} = maxH[t]$
      $\chi_2[t] = x_n[t]$ //posodobi najboljšo socialno komponento
      if $SC_s > 0$ **then**
        $SC_s = SC_s - 1$
      end if
      if $SC_s = 0$ **then** //podksupina je lahko nagrajena
        //preveri, če skupina lahko dobi novega agenta
        if $N_s < N_{max}$ in **Rand**$(0,1)\frac{1}{N_s^{kill}+1} >$ **Rand**$(0,1)$ **then**
          pošlji povpraševanje po agentu iz skupine izključenih
          if $N_s^{kill} > 0$ **then**
            $N_s^{kill} = N_s^{kill} - 1$ //pomanjšaj števec izključenih agentov
            if **Rand**$(0,1)\frac{N_s}{N_{max}} >$ **Rand**$(0,1)$ **then** //lahko skupina ustvari novo podskupino?
              pošlji povpraševanje za novo podskupino vsem agentom v skupini izključenih
              if $N_s^{kill} > 0$ **then**
                $N_s^{kill} = N_s^{kill} - 1$ //pomanjšaj števec izključenih
              end if
            end if

end if
end if
end if
else//podskupina se ni izboljšala
$SC_s = SC_s + 1$
if $SC_s = SC_{max}$ **then** //kaznuj podksupino
if $N_s > N_{min}$ **then** //preveri, če je mogoče izključiti najslabšega agenta
$N_s^{kill} = N_s^{kill} + 1$ //povečaj števec izključenih agentov
$SC_s = SC_{max}\left[1 - \frac{1}{N_s^{kill}+1}\right]$ //resetiraj števec napredka
if $h_{best} = minH[t]$ **then** //najslabši agent v podskupini
$swarmID = 0$ //izloči tega agenta
end if
else//razpusti celotno podskupino
$swarmID = 0$ //izloči tega agenta
end if
end if
if $g(x_n[t]) \geqslant g_{best}$ **then** //maksimiziraj razdaljo od ovir
$g_{best} = g(x_n[t])$
$\chi_3[t] = x_n[t]$
end if
$v_n[t+1] = w_n[t] + \sum_{i=1}^{3} \rho_i r_i(\chi_i[t] - x_n[t])$
$x_n[t+1] = x_n[t] + v_n[t+1]$
end if
else//agent je v skupini izključenih
naključno se sprehajaj po prostoru
oceni rešitev agenta $h_n[t]$
if $h_n(x_n[t]) > h_{best}$ **then** //agent se je izboljšal
$h_{best} = h(x_n[t])$
end if
Pošlji rešitev $h_n[t]$ in $x_n[t]$ ostalim agentom v podskupini, prejmi njihove informacije.

Zgradi vektor H[t], ki vključuje rešitve vseh agentov znotraj podskupine izključenih ($swarmID = 0$).
if $maxH[t] > H_{best}$ **then**
$H_{best} = maxH[t]$
if $h_{best} = max_{n_I} H[t]$ **then** //eden izmed najboljših N_I agentov v skupini izključenih
//ali lahko ustvarijo novo podskupino?
if $N_X \geqslant N_I$ in **Rand**$(0,1)\frac{N_X}{N_T} >$ **Rand**$(0,1)$ **then**
$swarmID = swarmID_{nov}$ //dodaj agenta v novo skupino
povprašaj po $N_I - 1$ agentov v skupini izključenih
else
if agent prejme informacijo o potrebi novega člana podskupine **then**
$swarmID = swarmID_{prejet}$ //dodaj agenta v skupino, ki ga potrebuje
$N_s = N_s + 1$
obvesti skupino o novi vrednosti N_s
end if
if agent prejme informacijo o potrebi kreiranja nove podskupine **then**
$swarmID = swarmID_{nov}$ //dodaj agenta v skupino, ki ga potrebuje
$N_s = N_I$ //ponastavi število agentov v skupini
$N_s^{kill} = 0$ //ponastavi števec izključenih agentov
$SC_s = 0$ //ponastavi vrednost števca
end if
end if
end if
end if
until zaustavitveni kriterij oz. konvergenca

Poglavje 4

Simulacijsko okolje Gridland

Simulacijsko okolje Gridland je zasnoval asistent magister Luka Čehovin, namen okolja pa je simulacija večagentnih sistemov, ki v prostoru tekmujejo za pridobitev virov (zastavic). Agenti so v okolju razdeljeni v več skupin, ki s komuniciranjem poskušajo pridobiti čim več zastavic in jih dostaviti v svojo bazo. Pri tem jih lahko nasprotne skupine agentov ovirajo in jim preprečijo zbiranje zastavic. Ob koncu je uspešnejša skupina agentov tista, ki zbere več zastavic.

4.1 Ustreznost okolja Gridland

Okolje Gridland upošteva kriterije navedene v Tabeli 2.2 za razvoj okolja večagentnih sistemov.

Gridland ima vgrajene protokole za komunikacijo in omogoča pošiljanje sporočil po načinu točka-v-točko. En agent lahko torej pošlje sporočilo drugemu agentu. Agent s strežnikom komunicira preko specificiranih sporočil in tako dobi podatke o okolju. Za razreševanje konfliktov skrbi sinhronizacija pri izvajanju metod, kar je posebno pomembno zaradi okolja, ki temelji na

modelu strežnik-odjemalec. Agenti imajo dodeljene unikatne indentifikacijske številke, njihove akcije pa se beležijo s časovnimi oznakami v zunanji datoteki.

Za povezavo agenta z okoljem skrbijo protokoli za komunikacijo med strežnikom in odjemalcem (agenti). Tako agent lahko dobi znanje o svoji okolici in o okolju poizveduje. Agent lahko pridobi podatke o zasedenosti polj in tako predvidi ovire in ostale agente, ki v okolju sobivajo. Okolje agentu omogoča realno-časovno poizvedovanje. Agent okolja ne more spreminjati in nima nadzora nad okoljem in ostalimi agenti.

Vsi agenti so zmožni razpoznati in pošiljati sporočila ter glede na prejeto znanje koordinirati svoje akcije in obnašanje. Vsi agenti iste skupine stremijo k istemu cilju in so med seboj enaki, zato vsi agenti poznajo modele ostalih agentov skupine.

4.2 Prilagoditev okolja Gridland

V originalni različici okolje ni bilo namenjeno preiskovanju celotnega prostora, zato je bilo potrebno okolje prilagoditi. Ker pri preiskovanju prostora ne gre za nabiranje virov, smo iz okolja odstranili zastavice. Za namene preiskovanja celotnega prostora smo omogočili vpogled v zgodovino vseh agentov, kar omogoča dodatno analizo gibanja agentov, kot sistema. Prav tako vpogled v zgodovino omogoča spremljanje že preiskanega (obiskanega) prostora. Za merjenje uspešnosti algoritmov smo implementirali tudi zaznavanje pokritosti celotnega prostora.

4.2.1 Zaznavanje cilja algoritmov

Praviloma se algoritmi za preiskovanje zaključijo, ko agentom uspe preiskati celoten prostor. Zaradi narave nekaterih algoritmov agenti nikoli ne uspejo preiskati celotnega prostora in je njihov cilj preiskati le določen delež. V okolju Gridland smo omogočili zaznavanje pokritosti prostora in simulacija se ustavi, ko agenti preiščejo celoten prostor ali pa delež prostora. Kot preiskan prostor se šteje samo prostor, ki ga je agent dejansko obiskal.

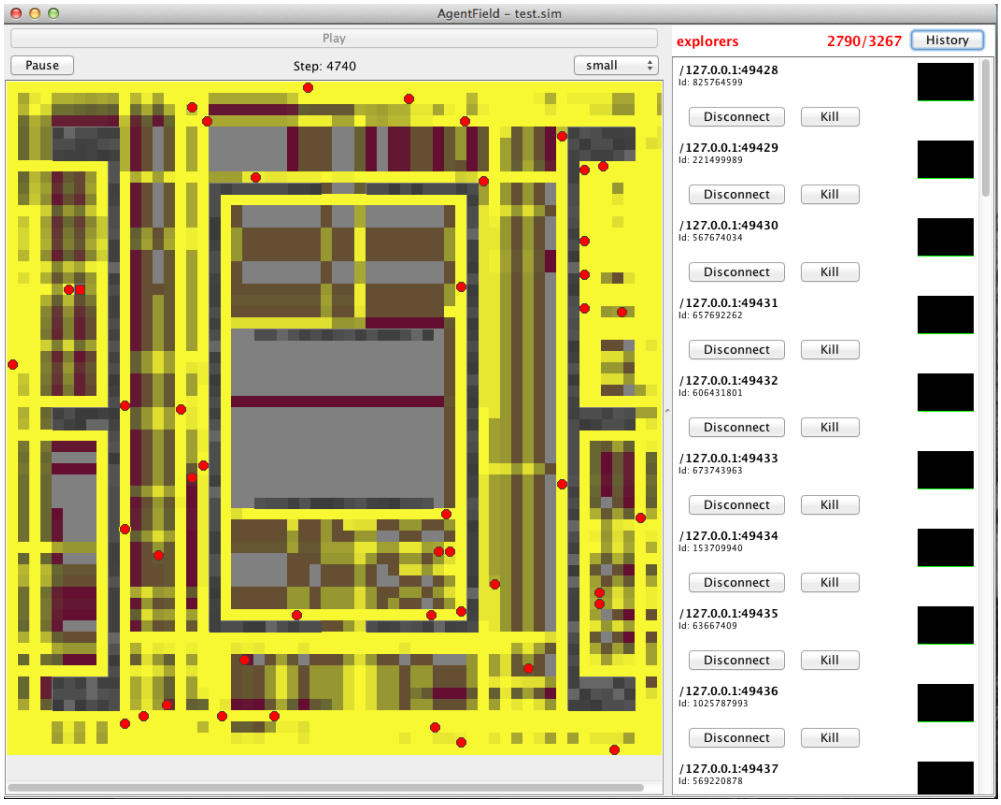
4.2.2 Skupna zgodovina agentov

Okolje Gridland je že omogočalo vpogled v zgodovino posameznega agenta. Za potrebe preiskovanja prostora z večimi agenti pa smo funkcionalnost razširili še na izbrano množico agentov. Tako lahko dobimo vpogled v zgodovino celotne skupine agentov, ki preiskuje prostor. To omogoča tudi spremljanje že preiskanega prostora in kasnejšo analizo za določanje uspešnosti algoritmov, saj lahko ugotovimo na katerih delih prostora so se agenti zadrževali dlje časa. Ob vpogledu v zgodovino se celice obarvajo s svetlejšo barvo, če so bile večkrat obiskane. V primeru, da agent še ni dosegel polja, vendar se nahaja v njegovi okolici, je polje obarvano s sivo barvo. Največkrat obiskane celice se obarvajo z rumeno barvo, najmanjkrat pa s temno rdečo.

Primer vpogleda v zgodovino skupine agentov je viden na Sliki 4.1.

4.2.3 Izpis obiskanih koordinat

Gridland-u je dodana tudi možnost izpisa vseh obiskanih koordinat za celotno skupino. To omogoča kasnejšo analizo premikanja agentov. S pomočjo tega lahko razberemo kje se je agent zadrževal in pa tudi koliko premikov je skupina naredila glede na določen procent preiskanega prostora.



Slika 4.1: Vpogled v zgodovino celotne skupine agentov.

Okolje Gridland je z opisanimi prilagoditvami pripravljeno za implementacijo, primerjavo in analizo različnih algoritmov za porazdeljeno preiskovanje prostora. V naslednjem poglavju (5) so opisane implementacije treh različnih agentov, v poglavju 6 pa so opisani rezultati in podrobna analiza algoritmov.

Poglavje 5

Implementacija algoritmov za porazdeljeno preiskovanje prostora

Implementirali smo tri različne algoritme. Prvi je naključni algoritem, ki predstavlja referenčno oceno za preiskovanje prostora. Drugi je predstavnik "pseudo-naključnih" algoritmov, kjer agenti upoštevajo določena pravila in, ko jim ta ne služijo več, smer izberejo naključno. Tretji pa je eden izmed najbolj naprednih in najnovejših algoritmov na področju optimizacije roja delcev. Ta algoritem je namenjen iskanju ciljev v prostoru, vendar smo se odločili algoritem prilagoditi in preizkusiti za namene preiskovanja prostora. Kako se je algoritem izkazal, je opisano v poglavju 6, kjer so podrobneje predstavljeni rezultati.

5.1 Naključni algoritem

Za referenčno oceno potrebnega časa preiskovanja prostora z večimi agenti smo razvili agente, ki naključno preiskujejo prostor (Algoritem 6). Agenti med seboj ne komunicirajo in se ne zavedajo katere dele prostora so že

preiskali (si ne izmenjujejo zemljevida prostora). Vsaka njihova odločitev je naključna, njihove zmožnosti pa so omejene na izogibanje oviram, izogibanje ostalim agentom in premikanje po prostoru.

Agenti se gibljejo po prostoru tako, da se v vsakem koraku odločijo za novo smer. Na voljo imajo štiri smeri (gor, dol, levo ali desno), ki se zapišejo v tabelo. Smeri zapisane v tabelo se nato premešajo z algoritmom Fisher-Yates[1]. Ko je tabela premešana, agent od strežnika pridobi podatke o okolju in se nato odloči za prvo smer v tabeli, ki je mogoča.

5.2 Pseudo naključni algoritem

Ker so se pseudo-naključni algoritmi izkazali kot uspešni (npr. [4]), smo kot primer implementirali tudi lasten pseudo-naključen algoritem. Tudi tokrat agenti med seboj ne izmenjujejo podatkov in se ne zavedajo že preiskanega prostora. Za razliko od naključnih agentov pa se držijo določenih pravil. Podroben opis Pseudo-naključnega agenta je zapisan z Algoritmom 7.

Pravila, ki jih agent upošteva so naslednja

1. izogibaj se oviram in ostalim agentom
2. premikaj se v izbrani smeri, dokler ne naletiš na oviro
3. ko naletiš na oviro, izberi novo naključno smer
4. možnosti za novo smer izključujejo nasprotno smer, razen če je le-ta edina možna
5. vsakih n korakov naključno spremeni smer

[1] ang. "Fisher-Yates shuffle" oziroma "Knuth shuffle" zgenerira naključno permutacijo elementov končne množice.

Algoritem 6 Naključni agent

```
1: smerPremika = [levo, desno, gor, dol]
2: repeat
3:    premešajTabelo(smerPremika)
4:    for i = 0; i < smerPremika.length(); i++) do
5:       if moženPremik(smerPremika[i]) then
6:          premakniAgenta(smerPremika[i])
7:          break;
8:       end if
9:    end for
10: until zaustavitveniKriterij
11:
12: //Fisher-Yates shuffle
13: Function premešajTabelo( smerPremika )
14: for (i = 0; i < (n − 1); i++) do
15:    j = RandInt(0, i)
16:    if j ≠ i then
17:       a[i] ← a[j]
18:    end if
19:    a[j] ← smerPremika[i]
20: end for
```

Algoritem 7 Pseudo-Naključni agent

```
1: repeat
2:   smerPpremika = [levo, desno, gor, dol]
3:   a = RandInt(minKorakov, maxKorakov)
4:   if stevecKorakov > a ali prviPremik then
5:     novaSmer = izberiNakljucnoSmer() //glej Algoritem 6
6:     premakniAgenta(novaSmer)
7:     stevecKorakov = 0, prviPremik = false
8:   else
9:     if moženPremik( trenutnaSmer ) then
10:      premakniAgenta( trenutnaSmer )
11:    else
12:      if ediniMoženPremik(nasprotnaSmer) then
13:        premakniAgenta(nasprotnaSmer)
14:      else//simbol \ pomeni razliko množic
15:        smerPremika = smerPremika \ nasprotnaSmer
16:        novaSmer = izberiNakljucnoSmer() //glej Algoritem 6
17:        premakniAgenta(novaSmer)
18:      end if
19:    end if
20:    stevecKorakov++;
21:  end if
22: until zaustavitveniKriterij
```

5.3 RDPSO

Kot primer algoritma, ki temelji na optimizaciji roja delcev, smo se odločili preizkusiti algoritem RDPSO. Je nadgradnja osnovne optimizacije roja delcev z zmožnostjo izogibanja oviram in preprečevanjem zatikanja v lokalnih optimumih s pomočjo evolucije podskupin celotnega roja.

V osnovi algoritem ni namenjen preiskovanju prostora, temveč iskanju ciljev v prostoru. Za reševanje problema porazdeljenega preiskovanja prostora, smo algoritem morali prilagoditi.

5.3.1 Diskretiziranje prostora

Ker je algoritem namenjen reševanju problemov v zveznem prostoru, je potrebno prilagoditi funkcijo, ki izračuna novo pozicijo agenta tako, da vrne pozicijo v diskretnem okolju. Okolje Gridland agentovo pozicijo spremlja relativno glede na točko izvora, zato mora biti funkcija zmožna vračati tudi negativna števila.

Primer: Če se agent pomakne za dve polji levo in eno polje nižje od točke izvora, so njegove nove koordinate enake $\{-2, -1\}$.

5.3.2 Točka izvora

V osnovni obliki algoritma optimizacije roja delcev, se delci v prostoru inicializirajo naključno. Tako vsak delec predstavlja potencialno rešitev.

Okolje Gridland agente inicializira v eni izmed štirih točk okoli točke izvora (ang. “headquarters”, prikazana na Sliki 5.1), kar je bolj realistično v realnih problemih. To predstavlja velik problem za algoritem, saj je ocena rešitev agentov za vse agente približno enaka. Da bi agenti imeli različne rešitve

Slika 5.1: Točka izvora - označena s kvadratom.

že ob inicializaciji algoritma, se prvih nekaj korakov premikajo naključno okoli točke izvore. Naključno premikanje traja dokler si vsi agenti med seboj ne izmenjajo informacije o članstvu v podskupinah. Ko je ta informacija sinhronizirana med vsemi agenti, pričnejo z izvajanjem algoritma.

5.3.3 Funkcija uspešnosti agenta

Funkcija uspešnosti pove kako dobra je trenutna pozicija agenta. V primeru iskanja ciljev, funkcija uspešnosti pove kako blizu cilja je agent. Za preiskovanje prostora je potrebno uporabiti drugačno funkcijo, kot v primeru iskanja ciljev.

Primer prvotnega problema: Če v prostor postavimo dve svetilki, ki svetita z različno intenziteto (ena temnejša in druga svetlejša) lahko za funkcijo uspešnosti uporabimo meritev senzorja za intenziteto svetlobe. Tako lahko agenta usmerjamo k cilju.

Ker algoritem uporabljamo za preiskovanje prostora, je agenta treba nagraditi za odkriti prostor in ga kaznovati, če predolgo ne odkrije novega prostora.

Funkcija kot vhodni podatek dobi razliko odkritega prostora od prejšnje ocene uspešnosti agenta, do trenutne ocene. Agent si shrani tudi prejšnji rezultat, ki vpliva na novi rezultat. Razlika odkritega prostora se izračuna s pomočjo beleženja zgodovine premikov agenta. Par koordinat (x, y) vsakega premika se shrani v množico Hist_n, v kateri so vsi elementi unikatni, zato

se isti par ne more zapisati več kot enkrat. Vsak par koordinat v množici tako predstavlja obiskano celico prostora, ko je ta prvič odkrita. Razlika preiskanega prostora (5.1) je tako razlika med velikostjo množice $Hist_n$ od ocene funkcije uspešnosti v času (t) in ocene funkcije uspešnosti v času $(t-1)$. Izračunane razlike preiskanega prostora se hranijo v seznamu Δ. Indeks zadnjega elementa seznama je označen s $\mathbf{k}$, ki je enak številu elementov v seznamu. To omogoča dostop do poljubnega števila zadnjih dodanih razlik v seznam, kar je upoštevano pri izračunu bonusa (5.3) in malusa (5.4). $\mathbf{a}$ predstavlja število upoštevanih preteklih razlik odkritega prostora v zgodovini agenta, ki se pri izračunu upoštevajo (uporabimo torej zadnjih a elementov iz seznama Δ).

$$\Delta(\mathbf{k}) = |Hist_n(t)| - |Hist_n(t-1)| \tag{5.1}$$

Funkcija je definirana kot,

$$h_n(t) = h_n(t-1) + b(t) - m(t) \tag{5.2}$$

kjer je $b(t)$ bonus in se izračuna kot (5.3)

$$b(t) = \sum_{i=0}^{\mathbf{a}} 2^{-i} - 2^{-i} * \Delta(\mathbf{k} - (i+1)) \tag{5.3}$$

$m(t)$ pa je malus in se izračuna kot (5.4),

$$b(t) = \sum_{i=-1}^{\mathbf{a}} 2^{-i} * \Delta(\mathbf{k} - (i+1)) \tag{5.4}$$

Funkcija uspešnosti (5.2) ima pri izračunu nove pozicije agenta velik pomen, saj vpliva na velikost območja (meje za relativni odmik po x in y koordinati od trenutne pozicije) v katerem agent izbira nove koordinate. V primeru, da funkcija hitro narašča se to območje zmanjša, saj to pomeni, da je v okolici

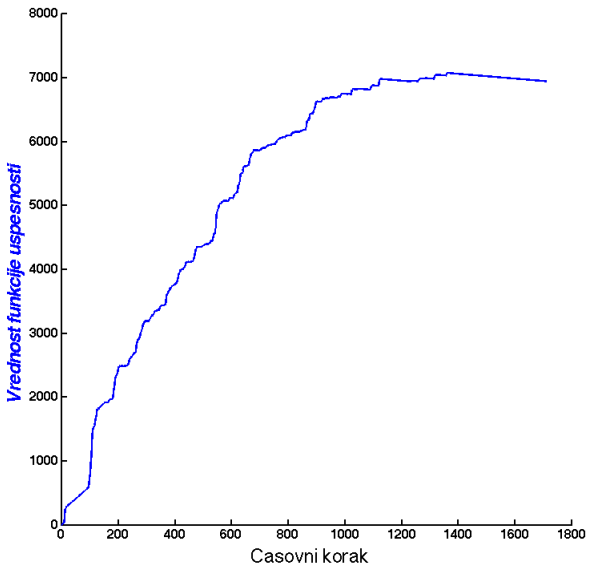
agenta še neodkriti prostor. Če funkcija hitro pada, pa se to območje veča, saj to pomeni, da je agent v svoji okolici odkril že večino neodkritega prostora. To agentu pomaga pri izogibanju lokalnega optimuma tako, da mu omogoči izbiranje novih koordinat dlje od lokalnega optimuma. Prav tako zmanjša možnost, da agent izbere koordinate, ki niso na zemljevidu (agent se namreč ne zaveda velikosti prostora, ki ga preiskuje).

Velikost območja se določi tako, da se izračuna diferenčni kvocient funkcije uspešnosti (5.5), ta se normalizira na interval med -1 in 1, z deljenjem trenutnega diferenčnega kvocienta funkcije uspešnosti z maksimalno absolutno numerično vrednostjo, ki jo lahko zavzame bonus (5.3) ali malus (5.4). Tako normaliziran diferenčni kvocient predstavlja faktor A (5.7), s katerim se pomnoži interval $[-a, m]$, kjer je $-a$ izračunan kot (5.6), ki predstavlja meje za relativni odmik od trenutne pozicije.

Glede na predznak faktorja A se loči dva primera. Če je faktor pozitiven se interval izračuna po (5.8), če pa je negativen se uporabi (5.9). Vrednost m predstavlja mejo intervala in se dinamično spreminja glede na odkrito velikost prostora. Če agent ugotovi, da je prostor večji od že poznanega prostora, lahko mejo m poveča. Tako lahko z izbiranjem koordinat obišče neodkrite dele prostora.

Faktor za določanje območja je naveden v (5.7). Agent lahko nato izbere x in y koordinato iz območja, ki ga določa izračunani interval. Za dejanski izračun koordinat na določenem območju se nato uporabi (3.1). Primer funkcije agenta, ki preiskuje prostor je viden na grafu na Sliki 5.2, kjer funkcija narašča dokler agent odkriva še neobiskan prostor. Po približno 1100 časovnih korakih vrednost preneha naraščati, ker agent odkriva vedno manj prostora in se približuje lokalnemu optimumu, kjer bo vrednost funkcije začela padati.

$$\mathtt{diff} = h_n(t) - h_n(t-1) \tag{5.5}$$



Slika 5.2: Funkcija uspešnosti agenta, ki narašča, dokler agent odkriva nov prostor.

$$-a = -m * \frac{min(A)}{max(A)} \tag{5.6}$$

$$A = \frac{h_n(t) - h_n(t-1)}{maxA} \tag{5.7}$$

$$(1 - |A|) * [-a; m] \tag{5.8}$$

$$|A| * [-a; m] \tag{5.9}$$

Funkcija uspešnosti za skupino izključenih

V kolikor so agenti člani skupine izključenih uporabijo drugačno funkcijo uspešnosti. Razlog za to je v tem, da se agenti v skupini izključenih po prostoru premikajo naključno. To jim prepreči izboljšanje vrednosti, če se zadržujejo na enem območju. Ko se skupina izključenih dolgo časa zadrži na ožjem območju, vrednost funkcije začne linearno padati. To preprečuje agentom v skupini izključenih, da bi se vključili v druge skupine, saj se to lahko zgodi le v primeru, da se skupina izključenih izboljšuje.

Funkcija uspešnosti za skupino izključenih je zato bolj prizanesljiva in agentom v primeru zadrževanja na ožjem območju odbija manjšo vrednost (faktorji v malusu so manjši). To vzpodbudi vključevanje v ostale skupine.

5.3.4 Težave z izbiranjem točk zunaj mape ali v nevidnem delu

Z namenom, da bi problem čim bolj približali resničnemu svetu se tudi v simulaciji agenti ne zavedajo kako velik je prostor, ki ga preiskujejo. Zato se lahko zgodi, da agent za novo pozicijo določi pozicijo v zaenkrat še ne videnem prostoru ali pa celo izven mape.

V tem primeru se agent postavi v raziskovalni način in se pomika proti obmejnim celicam, ki ločijo že viden del in neviden del prostora. Samo viden prostor se ne šteje kot preiskan, saj ga agent še ni obiskal.

5.3.5 Načini premikanja

Agent je zmožen delovati v enem izmed treh načinov premikanja:

1. naključno premikanje,
2. raziskovanje,

3. navigacija.

Naključno premikanje

Ob inicializaciji se agent premika naključno, dokler ne prejme potrebnih podatkov za sinhronizacijo tabele o članstvu z ostalimi agenti. Prav tako se agent, ki je član skupine izključenih, po prostoru premika naključno.

Raziskovanje prostora

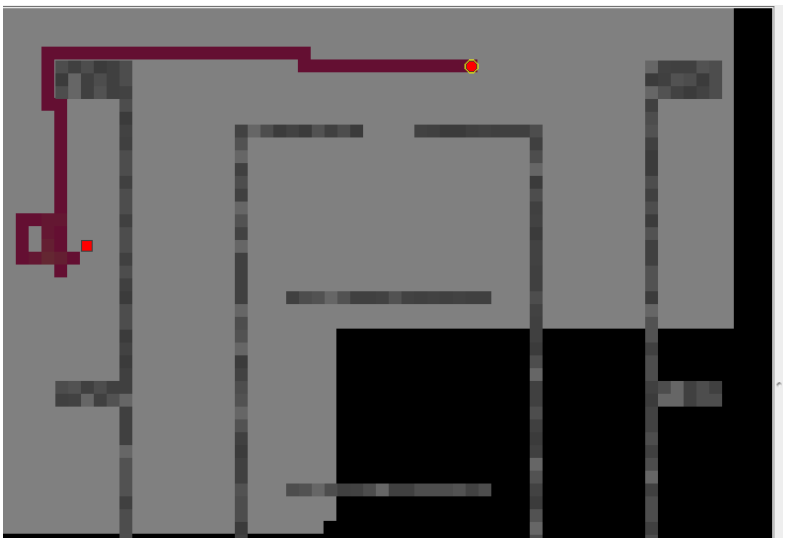
V kolikor agent izbere točko za katero ne ve ali je na zemljevidu ali ne, se posluži raziskovalnega načina. V tem načinu se agent premika do obmejnih celic (na Sliki 5.3 so obmejne celice tiste, ki so na robu med sivim in črnim prostorom). Raziskovalno premikanje se konča, ko agent v svoji vidni okolici zazna celico, ki jo mora obiskati. Če take celice ne zazna po tem, ko se je pomaknil do vseh obmejnih celic (v tem primeru bi bil na Sliki 5.3 celoten prazen in neobiskan prostor obarvan s sivo) in zaznal meje prostora ta način premikanja ni več mogoč. Ko se to zgodi, lahko agent uporablja lokalni zemljevid, ki si ga je zgradil v načinu raziskovanja, za navigacijo.

Navigacija po prostoru

Če agent za novo pozicijo izbere pozicijo, ki je v njegovem vidnem območju se premakne v način navigacije. Prav tako se ta način aktivira, po tem, ko je agent zaznal meje celotnega prostora.

Ob izbiri pozicije agent preveri, če se le ta nahaja na zemljevidu, ki si ga je zgradil. Če ugotovi, da pozicija obstaja in ni ovira, poišče najbližjo pot do izbrane celice s pomočjo algoritma Dijkstra[2] [8].

[2]Algoritem Dijkstra na danem grafu in podanem začetnem vozlišču odkrije najkrajše



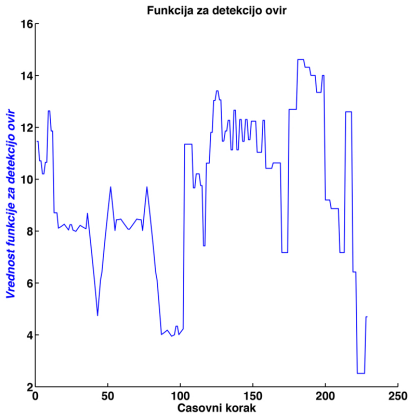
Slika 5.3: Raziskovanje prostora. Agent se pomika do obmejnih celic (na robu med sivim in črnim prostorom) in gradi lokalni zemljevid prostora.

Pot agenta do izbrane celice lahko prepreči srečanje s kakšnim drugim agentom. V tem primeru se agent ustavi in v naslednjem koraku zgradi nov načrt, ki upošteva zaznano oviro. Prav tako je nov načrt potreben, če agent še ne pozna meja celotnega prostora (še ni raziskal okolice po kateri se premika), saj mora upoštevati nove informacije o okolju.

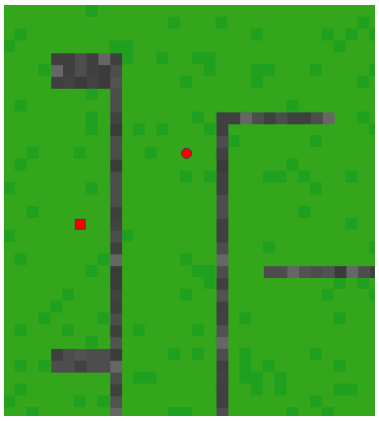
5.3.6 Izogibanje oviram

Izogibanje statičnim oviram algoritem doseže tako, da pri izračunu nove pozicije agenta upošteva tudi rezultat funkcije, ki izmeri razdaljo agenta do bližnjih ovir v okolju. Funkcija kot rezultat vrne seštevek Manhattenskih razdalj agenta do vseh ovir v njegovi vidni okolici. Vrednost funkcije za detekcijo ovir agenta v situaciji, ki je prikazana na Sliki 5.4b je prikazana na

poti od začetnega do ostalih vozlišč.



(a) Vrednost funkcije za detekcijo ovir.



(b) Prikaz agentove pozicije.

Slika 5.4: Prikaz vrednosti funkcije za detekcijo ovir (Slika 5.4a) in prikaz dela zemljevida, kjer so bile izvedene meritve (Slika 5.4b).

grafu na Sliki 5.4a.

5.3.7 Sinhronizacija informacije med agenti

Za uspešno delovanje algoritma je zelo pomembno, da so vrednosti, ki si jih agenti med seboj delijo, sinhronizirane.

Vsak agent ima svojo lokalno rešitev, ki jo posreduje svoji skupini. Skupina nato ustvari vektor, v katerem so zbrane vse rešitve agentov skupine. Ker vsak agent v skupini s pomočjo tega vektorja ugotovi ali se je skupina izboljšala ali poslabšala v trenutnem koraku (poišče maksimalno vrednost v vektorju) morajo rešitve v vektorju odražati čim bolj aktualno stanje vseh agentov.

V primeru, da se skupina ni izboljšala nekaj časa (kako dolgo, določa vrednost $\mathbf{SC_s}$), je skupino potrebno kaznovati tako, da se izključi najslabšega agenta v skupini. Agent mora tako ugotoviti ali je najslabši v skupini (poišče minimalno vrednost vektorja rešitev skupine) in preveriti, če je agentov v

skupini dovolj, da se lahko izloči. Če je v skupini ostal sam, pa mora skupino razpustiti. Zato je pomembno, da se agenti v vsakem koraku zavedajo kakšna je vrednost skupinskih spremenljivk, kot so $\mathbf{SC_s}$, število agentov v skupini in število obstoječih skupin.

Ker v porazdeljenih sistemih ni centralne entitete, ki bi nadzirala sinhronizacijo spremenljivk med posameznimi agenti, je sinhronizacija velik izziv. Vse skupaj oteži še omejitev razdalje za komunikacijo med dvema agentoma, kar pomeni da obstaja možnost, da agent nekaj časa ni zmožen komunicirati s svojo skupino, če se od nje preveč oddalji.

Spremljanje članstva agentov

Za pravilno delovanje vključevanja in izključevanja iz skupin je pomembno, da agenti vedo kateri skupini pripadajo in imajo to informacijo tudi o ostalih agentih. Za ta namen smo pripravili podatkovno strukturo, ki beleži članstvo vseh agentov.

Vsaka skupina je predstavljena z množico, v kateri je vsak podatek unikaten. Tako se prepreči možnost večkratnega dodajanja agenta v isto skupino, če pride do napake pri sinhronizaciji. Množice so nato shranjene v povezanem seznamu.

Vsak agent ima svojo tabelo članov in je nikoli ne zamenja s tabelo drugega agenta. Tabelo le posodablja z operacijami, ki jih zahtevajo ostali agenti, naštetimi v Tabeli 5.1. Za dodatno preprečevanje napak pri sinhronizaciji agent najprej prejme tabelo zahtevajočega agenta in na njegovi tabeli preveri, če je bilo izvedeno zaporedje operacij, ki se zahtevajo. Ko zaporedje zahtevanih operacij potrdi, agent izvede enako zaporedje na svoji tabeli.

Potrjevanje zahtevanih operacij je pomembno tudi v primeru, ko agent za svojo skupino zahteva novega agenta, ta pa dobi zahtevo po tem, ko je bila

skupina že izbrisana. S pomočjo potrjevanja agent, ki je prejel zahtevo za pridružitev skupini, ugotovi, da skupina ne obstaja več in tako zahtevo zavrne.

S pomočjo tabele je agent zmožen beležiti koliko članov je v njegovi skupini in koliko skupin obstaja. Tako ima vse potrebne podatke za pravilno delovanje vključevanja in izključevanja iz skupin ter ustvarjanje ali brisanje skupin.

Tabela 5.1: Operacije za beleženje članstva agentov.

vključi	vključi agenta v določeno skupino
izključi	izključi agenta iz določene skupine in ga vključi v skupino izključenih
izbriši	izbriše podskupino iz tabele
ustvari	doda novo podskupino v tabelo in vključi zahtevajočega agenta
združi	združi podatke ostalih agentov v svojo tabelo (ob inicializaciji)

Sinhronizacija sporočil

Za ohranjanje sinhronizacije pri komunikaciji smo razvili več tipov sporočil. To pripomore tudi k obvladovanju količine informacije, ki si jo agenti med seboj pošiljajo, kar prihrani sistemske vire. Vsi tipi sporočil so navedeni v Tabeli 5.2.

Za dodatno sinhronizacijo smo razvili sistem zahtev in potrditev. Tako lahko agenti zahtevajo določene akcije, ki pa se izvedejo le če so mogoče. V odgovor dobijo potrditev o obravnavi zahteve in lahko zahtevo umaknejo. S tem se prepreči večkratno vključevanje ali izključevanje agentov in večkratno ustvarjanje ali brisanje skupin.

Osnovno sporočilo (*info*) vsebuje tabelo o članstvu, vektor rešitev skupine,

spremenljivko SC_s in število izločenih agentov iz skupine. Je osnovno sporočilo, ki sinhronizira skupini pomembne podatke.

Izključevanje in vključevanje agentov v skupino je sinhronizirano preko sporočil "*agent_request*", "*agent_response*" in "*exclusion_request*". Prvi dve sporočili poskrbita za zahtevo in umik zahteve za potrebo po novem agentu. "*exclusion_request*" pa zahteva izključitev agenta iz skupine in posodobitev informacij o članstvu agentov za vse agente.

Ostala navedena sporočila poskrbijo za sinhronizacijo ustvarjanja in brisanja podskupin.

Tabela 5.2: Tipi sporočil za sinhronizacijo.

info	osnovno sporočilo za sinhronizacijo spremenljivk skupine in članstva
agent_request	sporočilo, ki zahteva vključitev novega agenta
agent_response	potrdi sprejeto zahtevo za vključitev novega agenta
subgroup_request	zahteva ustvarjanje nove skupine agentov
subgroup_response	potrdi sprejeto zahtevo za novo skupino
ni_request	zahteva priključitev $N_i - 1$ agentov v novo skupino
ni_response	potrdi zahtevo za priključitev v skupino
delete_broadcast	obvestilo o izbrisu skupine
exclusion_request	zahteva za izključitev agenta iz skupine

5.3.8 Izgradnja zemljevida

Agenti si med seboj delijo tudi podatke o že odkritem območju. Za izgradnjo zemljevida uporabljajo poseben tip sporočila, ki posreduje novo odkriti prostor ostalim agentom. To pripomore k hitrejši navigaciji do izbranih pozicij, saj agent lahko preveri ali pozicija obstaja na že videnem območju in ali je ta pozicija ovira ali ne.

Poglavje 6

Rezultati in primerjava algoritmov

V tem poglavju so predstavljeni rezultati simulacij preiskovanja prostora v okolju Gridland. Izvedene so bile simulacije za različne agente, z različnim številom agentov v skupini in na dveh različnih zemljevidih. Primerjali smo čas, ki so ga agenti potrebovali, da so preiskali določen delež prostora in kako je število agentov vplivalo na to. Ker je pri algoritmih za porazdeljeno preiskovanje prostora pomembna tudi učinkovitost, smo s pomočjo analize zgodovine premikov, med seboj primerjali učinkovitost različnih agentov.

6.1 Primerjava algoritmov

6.1.1 Število agentov

Simulacije za naključnega in pseudo-naključnega agenta so bile izvedene s skupinami velikosti 3, 6, 9 in 36 agentov. Za agenta RDPSO, se je 36 agentov izkazalo za preveč zahtevno, zato s skupino te velikosti ni bilo izvedenih simulacij. Izvedene so bile simulacije z eno podskupino in tremi agenti,

Random	3	6	9	36
Pseudo	3	6	9	36
RDPSO	3	6 $(2 * 3)$	9 $(3 * 3)$	/

Tabela 6.1: Število agentov v skupini.

z dvema podskupinama, kjer so bili v vsaki 3 agenti (skupaj 6) in s tremi skupinami, v vsaki po 3 agenti (skupaj 9). Število agentov v simulacijah je navedeno v Tabeli 6.1.

6.1.2 Parametri simulacije

Vse simulacije so se izvajale z enakimi parametri. Hitrost simulacije je bila nastavljena na 100, kar pomeni približno 6 sekund na 1000 korakov. Za beleženje števila korakov skrbi okolje Gridland in je neodvisno od agentov. Zaradi pošiljanja tabele članstva preko sporočil so bila sporočila v primeru velikega števila agentov večja od privzete dovoljene velikosti, zato je bil parameter nastavljen na vrednost 1024. Pošiljanje sporočil med agenti je bilo omejeno na oddaljenost agentov do 20 celic, med ponovnim pošiljanjem sporočila pa je moralo preteči vsaj 20 korakov simulacije. Agenti so se inicializirali v razmaku 30 strežniških korakov. Največje število agentov v simulaciji je bilo omejeno na 200. Skupina agentov se je imenovala *"explorers"*. Parametri in njihove vrednosti so navedene v Tabeli 6.2.

6.1.3 Izbira zemljevidov

Za primerna zemljevida smo izbrali dva, Beatle (Slika 6.1) in Parallel (Slika 6.2). Lastnosti prvega zemljevida vključujejo ograjo srednjega prostora, začetek na robu zemljevida in srednjo velikost celotnega prostora. Zemljevid Parallel pa ima začetek na sredini prostora, vendar vključuje prehode

simulation.speed	100
message.speed	20
message.neighborhood	20
message.size	1024
simulation.agents	200
simulation.respawn	30
team1	explorers

Tabela 6.2: Parametri simulacije.

med prostori, prav tako pa robove, kjer se agenti lahko potencialno zataknejo.

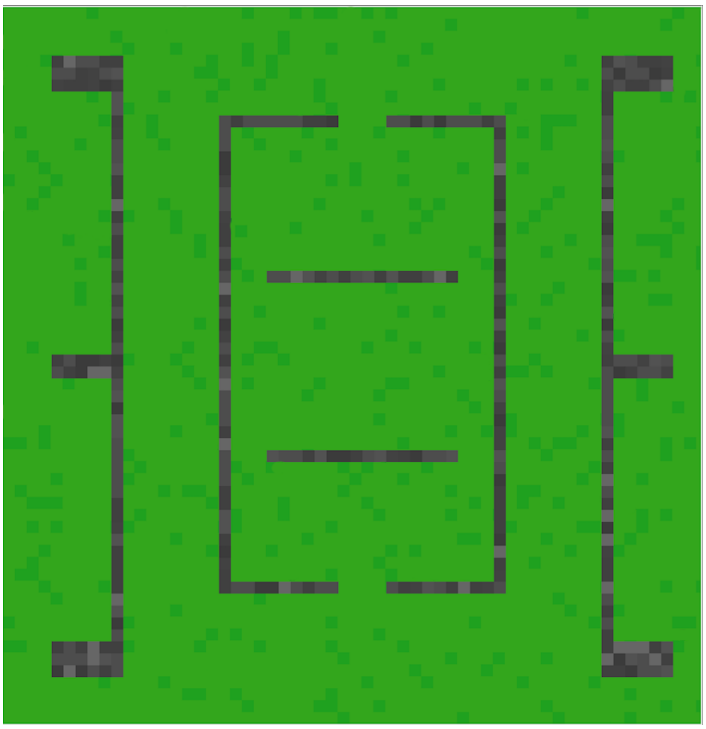
6.2 Rezultati

6.2.1 Primerjava potrebnega časa za preiskovanje prostora

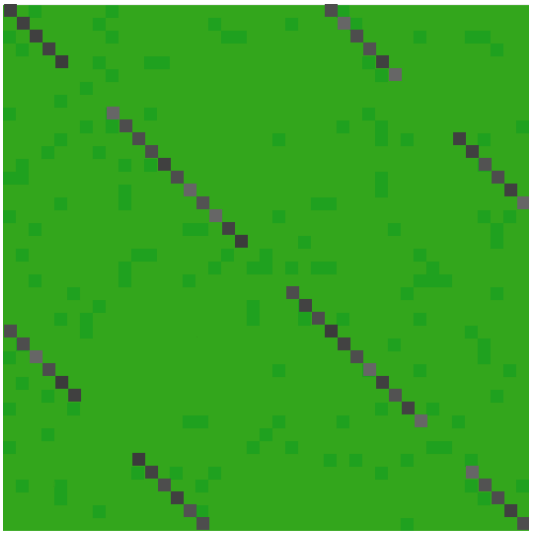
Za primerjavo časa potrebnega za preiskovanje prostora so bile izvedene simulacije, kjer so agenti preiskovali prostor, dokler niso preiskali 90% vsega prostora. Ko so ta cilj dosegli se je simulacija zaustavila. Razlog za omejitev cilja na 90% so težave pri odkrivanju zadnjih nekaj celic prostora. Večinoma so agenti za 100% odkriti prostor potrebovali več kot 450000 korakov.

Naključni agent

Iz grafov na Slikah 6.3a in 6.3b je razvidno, da naključni agenti za polovico preiskanega prostora potrebujejo občutno manj časa, kot za drugo polovico. Število agentov v skupini približno obratno sorazmerno zmanjša potreben čas za dosežen cilj. Očitna je tudi razlika med dvema zemljevidoma. Za-



Slika 6.1: Zemljevid Beatle.



Slika 6.2: Zemljevid Parallel.

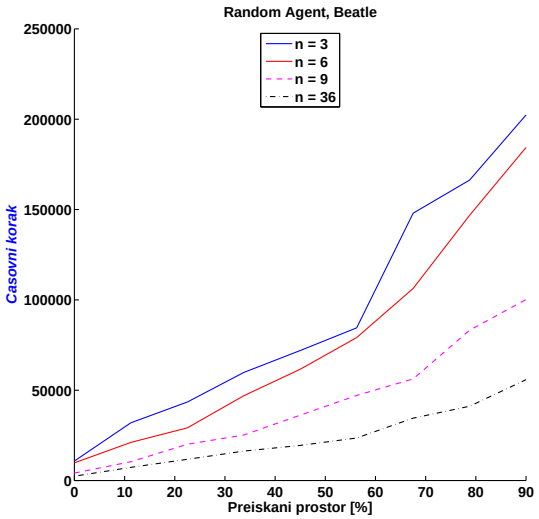
radi velikosti zemljevidov potrebnega časa ne moremo primerjati direktno, opazimo pa lahko, da zaradi večje zaprtosti zemljevida Beatle, agenti potrebujejo več časa na vsakih 10% preiskanega prostora. Na zemljevidu Parallel je težavnost preiskovanja bolj enakomerna, kljub naraščanju že odkritega prostora do 60%. Zemljevid Beatle predstavlja večjo težavnost.

Pseudo-naključni agent

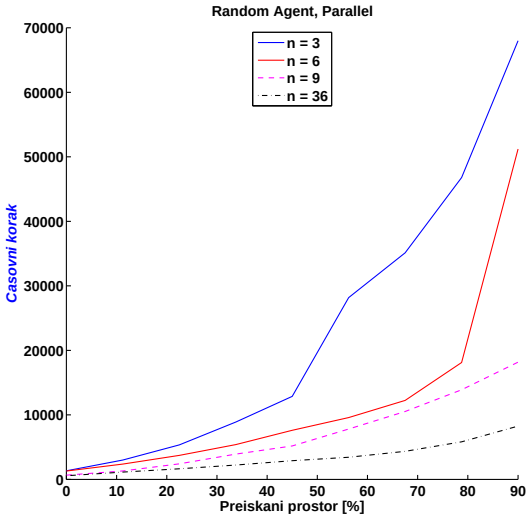
Največji problem Pseudo naključnega agenta je bilo zatikanje na robovih zemljevidov, kar je opazno na obeh zemljevidih. To je možno razbrati iz grafa na Sliki 6.4b, na grafu z zemljevidom Beatle (Slika 6.4a) pa to sicer ni tako izrazito in potreben čas na delež preiskanega prostora pada obratno sorazmerno s številom agentov. Hkrati pa je to omogočilo trke agentov v skupini, kar je povzročilo spremembo smeri. Zato ni presenetljivo, da je več agentov pripomoglo k boljšemu rezultatu, saj so večkrat spremenili smer.

Agent RDPSO

Agent RDPSO se ni najbolje izkazal, kar se tiče porabljenega časa za preiskovanje prostora. Opazno je počasno delovanje zaradi preobremenjenosti računskih virov, če je agentov več kot 6, zato je najboljše rezultate dosegala skupina šestih agentov, kjer so bili razdeljeni v dve podskupini po 3 agente (Sliki 6.5a in 6.5b). Ob simulacijah je opazno tudi, da agent RDPSO zaradi zahtevnosti algoritma več korakov stoji na mestu. Temu pripomore tudi dejstvo, da si mora agent vsakič, ko odkrije nov del prostora ponovno zgraditi načrt. Zaradi nezavedanja velikosti prostora pa si lahko izbere tudi koordinate, ki niso v prostoru, kar ga ponovno zaustavi za nekaj korakov.

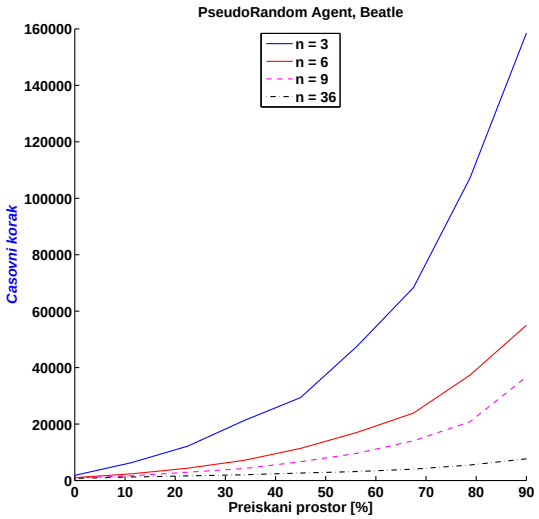


(a) Zemljevid Beatle z različnim številom naključnih agentov.

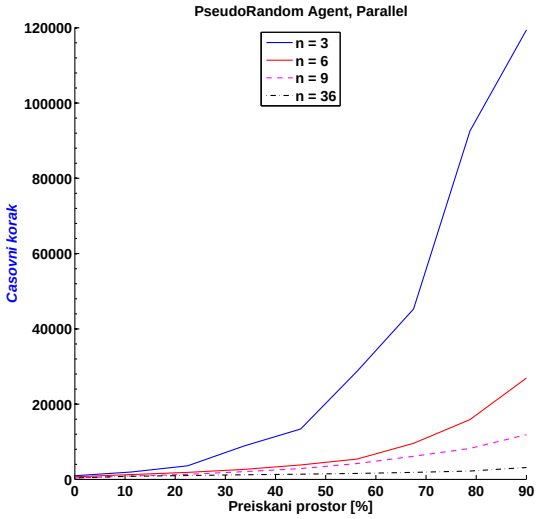


(b) Zemljevid Parallel z različnim številom naključnih agentov.

Slika 6.3: Zemljevid Beatle in Parallel z različnim številom naključnih agentov.

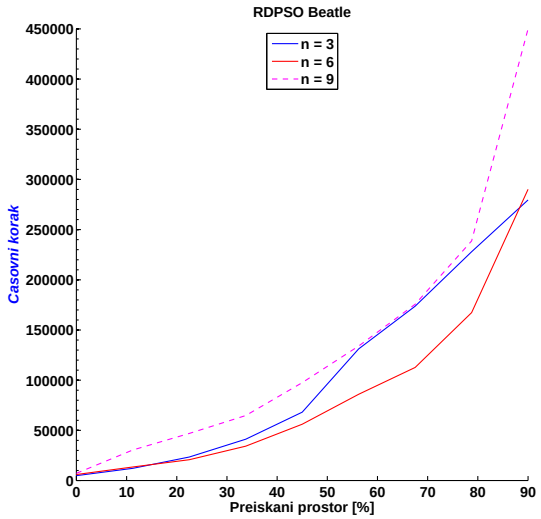


(a) Zemljevid Beatle z različnim številom agentov Pseudo.

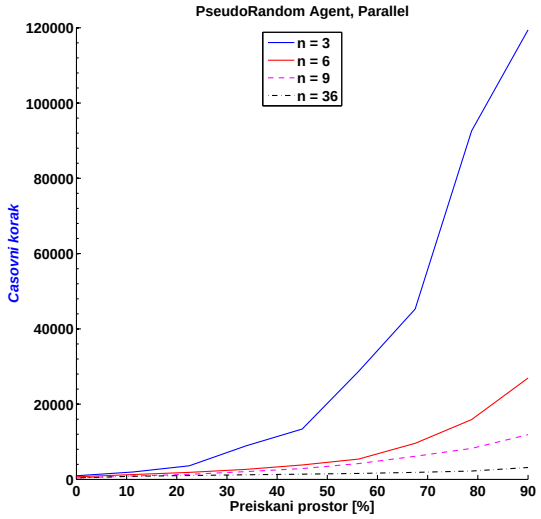


(b) Zemljevid Parallel z različnim številom agentov Psuedo.

Slika 6.4: Zemljevid Beatle in Parallel z različnim številom agentov Pseudo.

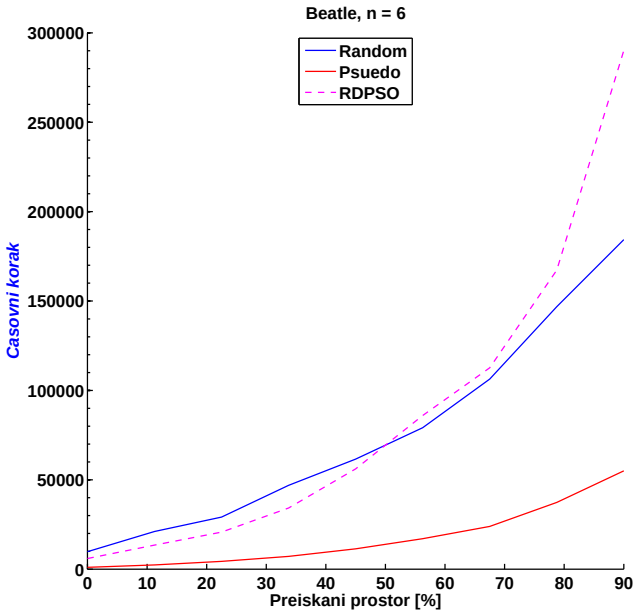


(a) Zemljevid Beatle z različnim številom agentov RDPSO.



(b) Zemljevid Parallel z različnim številom agentov RDPSO.

Slika 6.5: Zemljevid Beatle in Parallel z različnim številom agentov RDPSO.



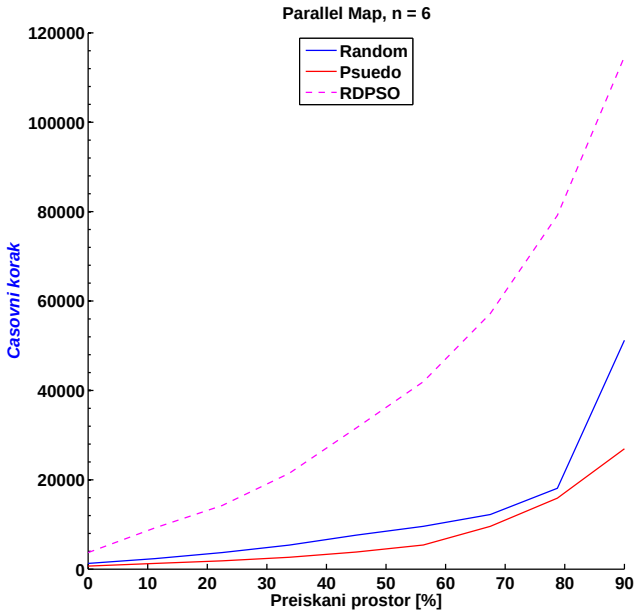
Slika 6.6: Zemeljevid Beatle z različnimi agenti, n = 6.

Zemljevid Beatle

Na zemljevidu Beatle (6.6) ima agent Pseudo najboljše rezultate. Agent RDPSO se je izkazal za boljšega od naključnega agenta, vendar z večjim deležem preiskanega prostora postane manj učinkovit, saj se zaradi podobne ocene funkcije uspešnosti agenti ne rešijo lokalnih optimumov in vedno več časa porabijo na enakih delih zemljevida.

Zemljevid Parallel

Zaradi večje odprtosti zemljevida Parallel (6.7) je hitrost odkrivanja prostora pri Pseudo in naključnemu agentu precej velika. Ker oba agenta veliko hitreje izračunata nove odločitve, je delež preiskanega prostora večji kot pri agentih RDPSO, ki potrebujejo veliko več časa za izračun novih koordinat. Pseudo



Slika 6.7: Zemeljevid Parallel z različnimi agenti, n = 6.

in naključni agent nimata težav z doseganjem različnih delov zemljevida in posledično do cilja prideta hitreje, kot pri zemljevidu Beatle.

6.2.2 Primerjava učinkovitosti agentov

RDPSO algoritem se na do sedaj videnih rezultatih ni najbolje obnesel. Razlogov za to je več. Eden izmed njih je ta, da podskupine ne pridejo do izraza, saj kljub inicializacijskemu postopku agenti ne začnejo na dovolj raznolikih koordinatah. Zato so vse podskupine še vedno podvržene zatikanju v lokalnih optimumih. Neuspešnosti glede na merjenje porabe časa za delež preiskanega prostora pripomore tudi počasno delovanje agentov. Agenti RDPSO potrebujejo več korakov za določitev novih koordinat in planiranje. Koraki na strani simulacije Gridland tečejo enako hitro za vse

agente, ne glede na to kako dolgo preračunavajo naslednjo potezo. Dodatna upočasnitev pa izhaja iz potrebe po ponovnem planiranju poti do izbranih koordinat, v primeru da agent posodobi poznani prostor ali mu pot do koordinat prepreči kakšen drug agent. Za razliko od agenta RDPSO se naključni agent in agent Pseudo za premik na novo mesto odločita v vsaki iteraciji (približno 10 korakov).

Na oceno algoritmov za preiskovanje prostora vpliva tudi učinkovitost agenta. S pomočjo okolja Gridland in zmožnosti izvoza zgodovine premikov agentov lahko ugotovimo koliko premikov je agent potreboval za delež preiskanega prostora. Če agente med seboj primerjamo po učinkovitosti, ugotovimo, da je agent RDPSO zelo učinkovit v primerjavi z naključnim agentom.

Analiza premikov po zemljevidu

Po končani simulaciji lahko iz okolja Gridland izvozimo zgodovino premikov po zemljevidu. V zgodovini se beležijo vse obiskane točke s strani agentov. Na grafih na Slikah 6.8 in 6.9 je prikazana frekvenca obiskanih koordinat. Večja kot je frekvenca, močneje je obarvana točka na grafu, ki predstavlja par absolutnih koordinat (x, y). Na vseh grafih je prikazana zgodovina premikov agentov po pretečenih 25000 korakih. Pomembno se je zavedati, da so v 25000 korakih agenti dosegli različno število premikov, kar je navedeno v Tabeli 6.3 in prikazano na grafih 6.10a in 6.10b. Zato je število točk na grafih, ki prikazujejo frekvenco obiskanih koordinat različno.

Iz grafov na Slikah 6.10a in 6.10b lahko razberemo, da je glede na število premikov v 25000 korakih agent Pseudo dosegel veliko višji delež preiskanega prostora na zemljevidu Beatle. Naključni agent in agent Pseudo imata na zemljevidu Parallel približno enako učinkovitost, razlog za to pa lahko razberemo iz grafa 6.9b. Razvidno je, da je agent Pseudo imel težave z zatika-

Agent	*Mapa*	*Št. Agentov (n)*	*Št. Premikov*	*Preiskani prostor* [%]
Random	Beatle	3	15539	18
Random	Beatle	6	31116	39
Random	Beatle	9	46591	49
Pseudo	Beatle	3	15630	48
Pseudo	Beatle	6	31158	75
Pseudo	Beatle	9	46714	81
RDPSO	Beatle	3	3425	17
RDPSO	Beatle	6	4138	23
RDPSO	Beatle	9	6606	32
Random	Parallel	3	15639	71
Random	Parallel	6	31138	86
Random	Parallel	9	46649	86
Pseudo	Parallel	3	15616	68
Pseudo	Parallel	6	31174	86
Pseudo	Parallel	9	46628	97
RDPSO	Parallel	3	2652	34
RDPSO	Parallel	6	5111	55
RDPSO	Parallel	9	4103	49

Tabela 6.3: Število premikov agentov v 25000 korakih.

njem v vogalih zemljevida. Slika 6.8b prikazuje zadrževanja agenta Pseudo na robovih zemljevida. Na grafu 6.8a pa lahko vidimo, da so naključni agenti zelo pogosto obiskali celice v okolici začetne točke.

Iz grafa na Sliki 6.8b je opazno, da so se agenti zadrževali na robovih. Več kot je agentov, več prostora so obiskali, saj večinoma spremembo smeri povzročijo trki z ostalimi agenti.

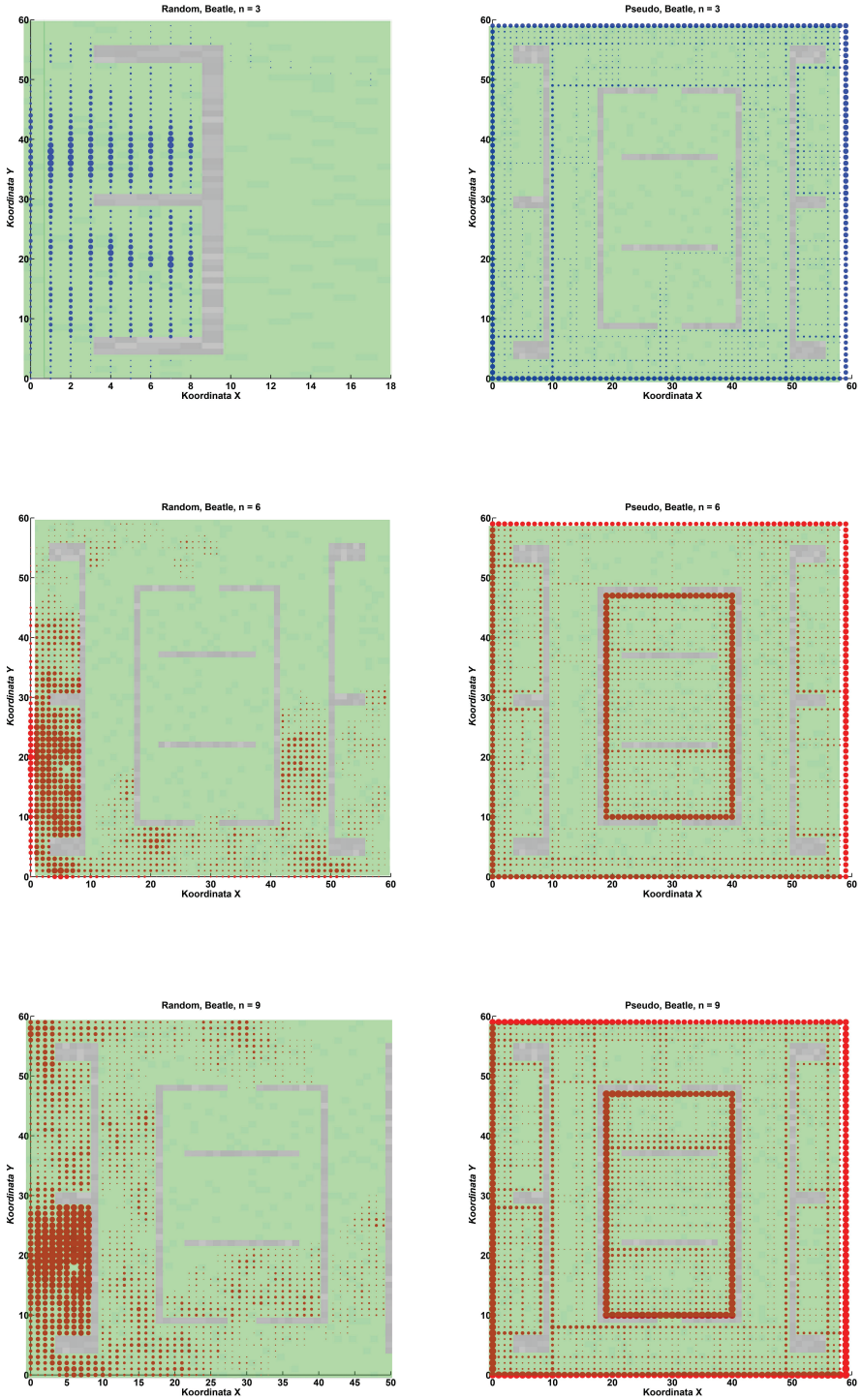
Najnižje število premikov doseže algoritem RDPSO, kar je razvidno na Sliki 6.9a. Premiki so za 3 in 6 agentov enakomerno porazdeljeni čez večji del zemljevida.

Iz grafov na Slikah 6.10a in 6.10b lahko intuitivno razberemo učinkovitost različnih agentov glede na število doseženih premikov v določenem številu korakov. Agent RDPSO naredi veliko manj premikov od naključnega in Pseudo agenta, vendar v enakem številu časovnih korakov preišče tudi manjši delež prostora. Naključni agent in agent Pseudo sta po številu premikov približno enakovredna, opazno pa je, da agent Pseudo preišče večji delež prostora. V primeru zemljevida Parallel (Slika 6.10b) je razlika v preiskanem deležu prostora manj izrazita.

Število premikov glede na delež odkritega prostora

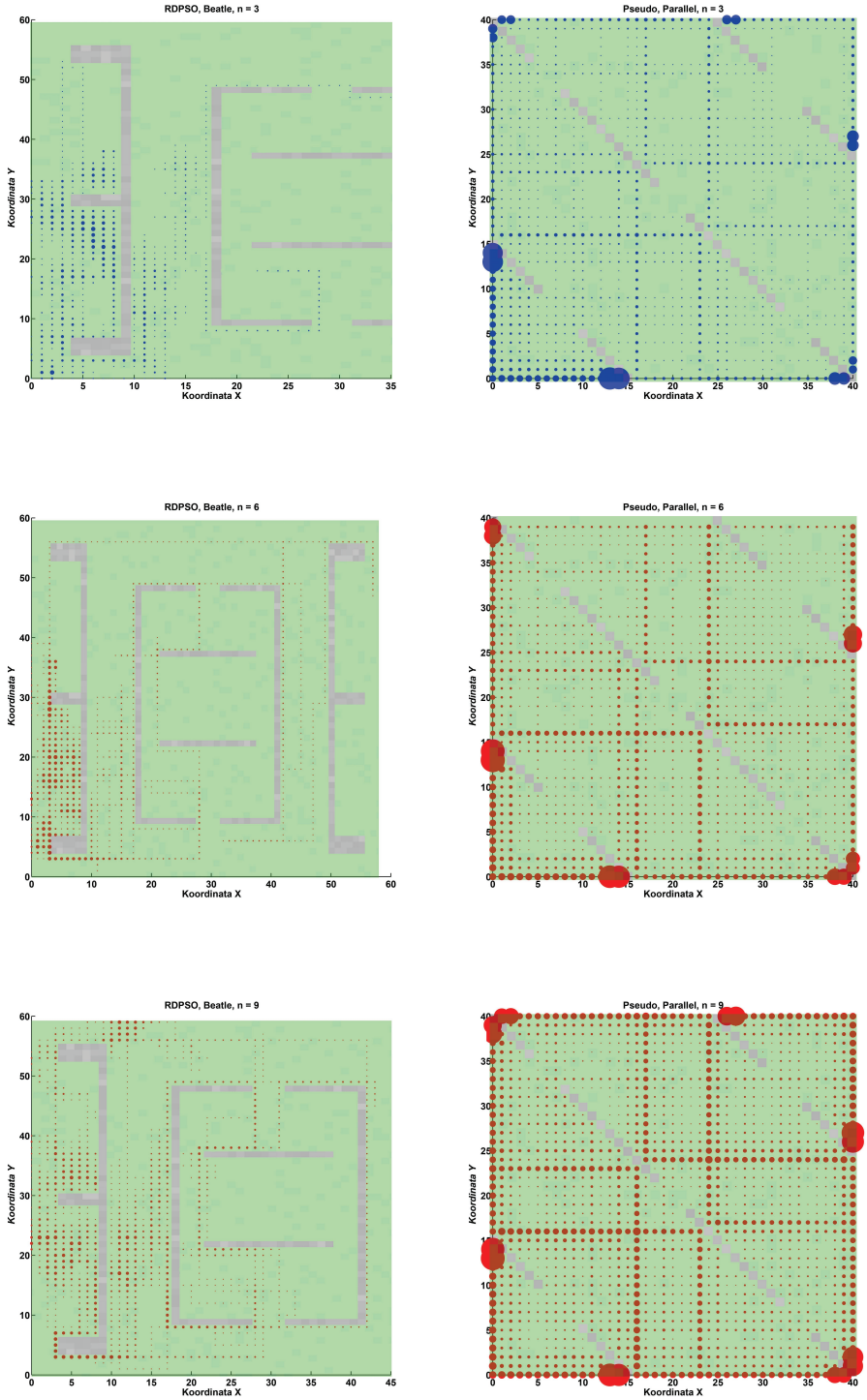
Grafa na Sliki 6.11 prikazujeta koliko premikov so potrebovali agenti glede na delež preiskanega prostora. V skupino je bilo vključenih 6 agentov. Agenti RDPSO so bili razdeljeni v dve podskupini, v vsaki pa so bili trije agenti. Na obeh zemljevidih sta agenta Pseudo in RDPSO bolj učinkovita od naključnega agenta.

Naključnim agentom zemljevidi odprtega tipa bolj ustrezajo, zato je njihova učinkovitost boljša na zemljevidu Parallel, kjer je razlika z agentom Pseudo manj izrazita. Opazno je, da je učinkovitost naključnih agentov pri prei-



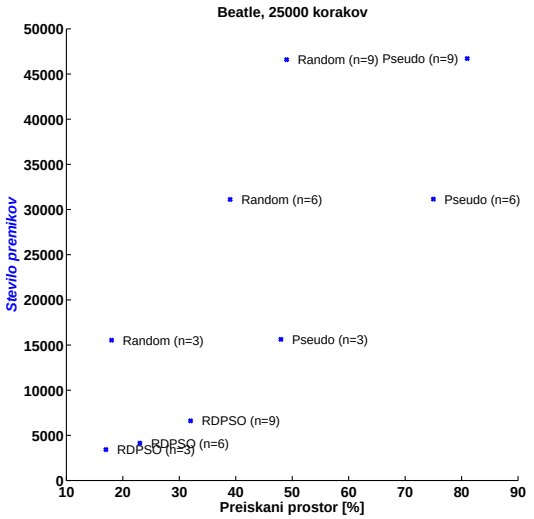
(a) Naključni agent **(b)** Agent Pseudo

Slika 6.8: Prikaz analize premikov za naključnega (6.8a) in agenta Pseudo (6.8b), na zemljevidu Beatle za 3, 6 in 9 agentov.

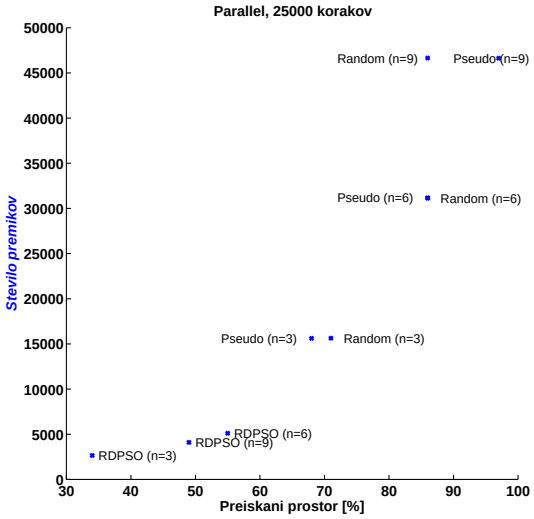


(a) Agent RDPSO, zemljevid Beatle. **(b)** Agent Pseudo, zemljevid Parallel.

Slika 6.9: Prikaz analize premikov za agenta RDPSO (6.9a) in Pseudo (6.9b), na zemljevidu Beatle in Parallel za 3, 6 in 9 agentov.



(a) Zemljevid Beatle.



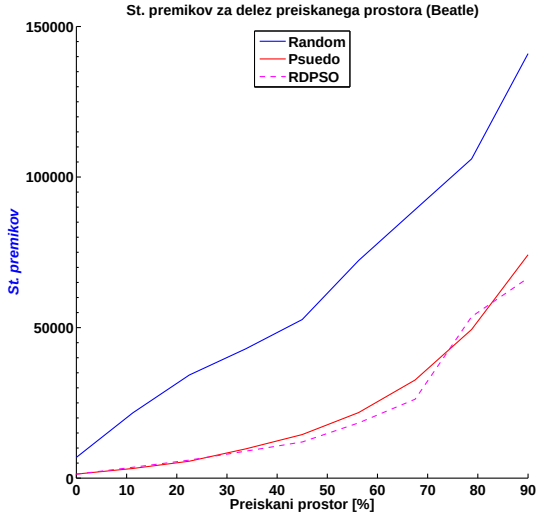
(b) Zemljevid Parallel.

Slika 6.10: Število premikov, glede na delež preiskanega prostora za zemljevid Beatle in Parallel, v 25000 korakih.

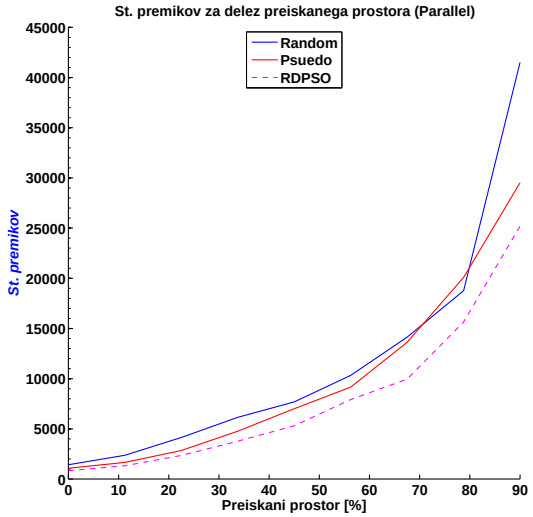
skovanju zadnjih 20% prostora veliko manjša. Po učinkovitosti je naključni agent slabši od agenta Pseudo, ker preišče manjši delež prostora in slabši od agenta RDPSO, ker za dosežen delež preiskanega prostora potrebuje veliko več premikov.

Agent RDPSO se na zemljevidu Parallel (Slika 6.11b) izkaže za bolj učinkovitega od agenta Pseudo, saj ima le ta glede na graf na Sliki 6.9b težave z zatikanjem v robovih zemljevida. Na zemljevidu Beatle razlika med agentoma RDPSO in Pseudo ni velika, po učinkovitosti sta primerljiva. Agenti RDPSO, ki so izključeni iz podskupine, uporabljajo naključni način premikanja in predstavljajo velik delež števila premikov vseh agentov v skupini. To pripomore k nižji učinkovitosti celotne skupine. Naključno premikanje agentov vpliva na učinkovitost predvsem na večjih zemljevidih, kot je zemljevid Beatle. Razlog za to je pomanjkanje komunikacije, zaradi prevelike oddaljenosti agentov, kar povzroči daljši čas za ponovno vključitev agenta v podskupino. Na zemljevidu Parallel se agenti RDPSO manj časa premikajo naključno, kar pripomore k večji učinkovitosti.

Čeprav se agent RDPSO ne izkaže za najboljšega, če primerjamo potreben čas na preiskan delež prostora, je po učinkovitosti boljši od tako naključnega kot tudi agenta Pseudo. Tako lahko zaključimo, da če bi agent RDPSO v danem času dosegel enako število premikov, kot ostala dva agenta, bi preiskal več prostora.



(a) Število premikov potrebnih za delež preiskanega prostora; zemljevid Beatle.



(b) Število premikov potrebnih za delež preiskanega prostora; zemljevid Parallel.

Slika 6.11: Prikaz potrebnega števila premikov za delež preiskanega prostora na zemljevidu Beatle in Parallel, za skupino 6 agentov.

Poglavje 7

Zaključek

7.1 Porazdeljeno preiskovanje prostora z večagentnim sistemom

Spoznali smo s kakšnimi problemi se ukvarja porazdeljena umetna inteligenca in kako nekatere izmed njih lahko rešimo s pomočjo večagentnih sistemov. Za učinkovito delovanje večagentnih sistemov potrebujemo tudi primerno okolje v katerem agenti lahko sobivajo. K uspešnosti večagentnih sistemov močno pripomorejo primerni protokoli za komunikacijo, interakcijo in koordinacijo.

Strategij za preiskovanje prostora je veliko, delijo pa se v večje skupine. Podrobneje smo spoznali algoritem za naključno preiskovanje, algoritme iz skupine pseudo-naključnih algoritmov, kjer se agenti držijo določenih pravil in pa algoritmov, ki spadajo v optimizacijo roja delcev. V tej skupini smo predstavili osnovni algoritem, prilagojen za soočanje z resničnim svetom, z možnostjo izogibanja oviram in pa njegovo nadgradnjo **RDPSO**, ki s pomočjo evolucije podskupin prepreči zatikanje v lokalnih optimumih.

Za simulacijo in podrobnejšo analizo algoritmov smo uporabili simulacijsko okolje Gridland, ki smo ga prilagodili za algoritme za porazdeljeno preiskovanje prostora. Implementirali smo tri različne agente. Za referenčne rezultate naključni algoritem in izboljšani pseudo-naključni algoritem, ki je predstavnik preprostih algoritmov brez koordinacije. Čeprav RDPSO v osnovni različici ni namenjen preiskovanju prostora, temveč iskanju ciljev v prostoru, smo algoritem prilagodili in ga preizkusili. Kljub težavam pri prilagoditvi algoritma na zastavljen problem, se je algoritem izkazal za uspešnega pri primerjavi učinkovitosti agentov. Pseudo-naključni agent se je odrezal veliko bolje od naključnega in agenta RDPSO po deležu preiskanega prostora. Zaradi trkov med agenti, ki so povzročili spremembo smeri pri agentu Pseudo, je bil delež preiskanega prostora še večji, če je bilo v skupini več agentov. Zaradi računske zahtevnosti agenta RDPSO je bilo preveliko število agentov manj učinkovito. Najboljše rezultate glede na učinkovitost so tako dosegli agenti RDPSO, ki so bili razdeljeni v dve podskupini, kjer so bili v vsaki po trije agenti.

Raziskave na področju preiskovanja prostora so zelo napredovale v zadnjih letih, vendar imajo večji del poti še pred seboj. Zaenkrat so roboti, ki so zmožni izvajati zahtevne algoritme, predragi in zelo pogrešljivi v primeru odpovedi sistema. Algoritmi, ki temeljijo na optimizaciji roja delcev, to težavo rešijo, vendar so večinoma učinkoviti le v simulacijah. Dobre rezultate dosegajo pri problemu iskanja ciljev v prostoru, za preiskovanje prostorov pa je potrebna še dodatna prilagoditev. Algoritem RDPSO je dober korak proti prilagoditvi na problem preiskovanja prostora in prenos optimizacije roja delcev v realni svet, vendar je potrebno rešiti še veliko težav.

Literatura

[1] Antoine Bautin, Olivier Simonin, François Charpillet, et al. Towards a communication free coordination for multi-robot exploration. In *6th National Conference on Control Architectures of Robots*, 2011.

[2] Wolfram Burgard, Mark Moors, Dieter Fox, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 476–481. IEEE, 2000.

[3] Wolfram Burgard, Mark Moors, and Frank Schneider. Collaborative exploration of unknown environments with teams of mobile robots. In *Advances in plan-based control of robotic agents*, pages 52–70. Springer, 2002.

[4] Chee Kong Cheng and Gerard Leng. Cooperative search algorithm for distributed autonomous robots. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 394–399. IEEE, 2004.

[5] Shu-Chuan Chu, John F Roddick, and Jeng-Shyang Pan. Parallel particle swarm optimization algorithm with communication strategies. *submitted to IEEE Transactions on Evolutionary Computation*, 2003.

[6] Micael S Couceiro. Evolutionary robot swarms under real-world constraints. 2013.

[7] Micael S Couceiro, Rui P Rocha, and Nuno MF Ferreira. A novel multi-robot exploration approach based on particle swarm optimization algorithms. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 327–332. IEEE, 2011.

[8] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[9] Russ C Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[10] Douglas W Gage. Randomized search strategies with imperfect sensors. In *Optical Tools for Manufacturing and Advanced Automation*, pages 270–279. International Society for Optics and Photonics, 1994.

[11] Alan H. Bond and Les Gasser. *A survey of Distributed artificial intelligence*. University of Southern California, 1988.

[12] James M Hereford. A distributed particle swarm optimization algorithm for swarm robotic applications. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1678–1685. IEEE, 2006.

[13] James M Hereford, Michael Siebold, and Shannon Nichols. Using the particle swarm optimization algorithm for robotic search applications. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 53–59. IEEE, 2007.

[14] Michelle McPartland, Stefano Nolfi, and Hussein A Abbass. Emergence of communication in competitive multi-agent systems: a pareto multi-objective approach. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 51–58. ACM, 2005.

[15] Yogeswaran Mohan and SG Ponnambalam. An extensive review of research in swarm robotics. In *Nature & Biologically Inspired Computing,*

2009. NaBIC 2009. World Congress on, pages 140–145. IEEE, 2009.

[16] NOAA. How much of the ocean have we explored?, 2014.

[17] Goncalo Pereira. Particle swarm optimisation. 2011.

[18] Jason Tillett, T Rao, Ferat Sahin, and Raghuveer Rao. Darwinian particle swarm optimization. 2005.

[19] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT press, 1999.

[20] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE, 1997.

[21] Brian Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents*, pages 47–53. ACM, 1998.

[22] Brian Yamauchi, Alan Schultz, and William Adams. Integrating exploration and localization for mobile robots. *Adaptive Behavior*, 7(2):217–229, 1999.